VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Systém pro sledování a reakci na bezpečnostní incidenty

Security Incidents Monitoring and Response System

2017 Bc. Roman Rotter

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Roman Rotter**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Systém pro sledování a reakci na bezpečnostní incidenty
Security Incidents Monitoring and Response System

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je navrhnout modulární systém s modulem pro centralizovaný monitoring antivirových alarmů z oddělených zákaznických prostředí spravovaných firmou Tieto, který je schopen automatizovaného rozhodování na základě získaných zkušeností a aktivního snižování režie potřebné k reakci na incident operátorem. Systém je nutné navrhnout tak, aby jej bylo možné rozšířit o další vstupy pokrývající bezpečnost IT infrastruktury (IPS, SIEM, Vulnerability management...), podporu IT procesů v Security týmu, moduly podporující / monitorující automatizační úlohy v cloudovém řešení a další. Systém musí zohledňovat aktuální trendy DevOps+ITIL modelu správy IT, kladoucí maximální důraz na automatizaci, rychlost a počet změn uváděných do praxe, transformaci okolních služeb, integraci dat, auditování, spolupráci na vývoji systému v rámci týmu s různým stupněm znalostí a zaměření.

1. Průzkum existujících řešení a jejich vlastností.
2. Analýza požadavků a návrh systémů.
3. Popis automatizovaných úloh a metod strojového učení použitých k jejich realizaci.
4. Návrh implementace systému.
5. Popis implementace a testování systému.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Přemysl Blahut**

Konzultant diplomové práce: doc. Ing. Jan Platoš, Ph.D.

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 26. dubna 2017 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 26. dubna 2017

.....

Tieto Czech s.r.o.
28. října 3346/91
702 00 Ostrava - Moravská Ostrava
IČO 64608051 DIČ CZ64608051

Na tomto místě bych rád poděkoval Mgr. Přemyslu Blahutovi a doc. Ing. Janu Platošovi, Ph.D. za odborné vedení, čas a cenné připomínky k vypracování této diplomové práce.

Dále bych rád poděkoval své rodině za trpělivost a podporu, kterou mi po celou dobu studia poskytovala.

Abstrakt

Tato práce řeší centralizovaný monitoring antivirových alertů pro firmu Tieto s podporou automatizace při reakci na jednotlivé incidenty. Systém je navržen modulárně s ohledem na rozšiřování o další problémové domény. Export a import dat je řešen Windows službami, podpora pro konkrétní typy dat a systémů je řešena formou pluginů. Přenosová vrstva je volitelná, primárně se využívá vlastní MessageBus a SMTP. Grafické rozhraní je řešeno WPF desktopovou aplikací nad frameworkem PRISM a DI Unity. První vrsta automatizace redukuje podobné alerty pomocí agregace a ztrátové komprese a následně pomocí strojového učení rozhoduje, které alerty vyžadují další eskalaci. Druhá vrstva automatizace je založena na systému pravidel s vlastní gramatikou, šablonovacího engine a rozhraní pro integraci s externími systémy. Navrhne instrukce k řešení daného problému, vyhledá kontakty a pomocí integrace vytvoří požadavek k řešení problému. Systém je možné přizpůsobit vždy aktuálním procesům, okolním systémům a síťové infrastruktuře. S minimálními náklady dovede ušetřit tisíce člověkohodin ročně, při celkovém zvýšení kvality nabízených služeb.

Klíčová slova: Antivirový monitoring, Automatizace, Strojové učení, WPF, WCF, PRISM, Unity, Dependency Injection, MEF, Windows Service, ANTLR, C#, R

Abstract

The target of this thesis is designing and implement centralized monitoring of antivirus alerts for Tieto Company with automation support for incident response. The system is designed as modular architecture with extensibility support. Export and import of data are implemented as Windows services which loads appropriate plugins for target systems. Transfer layer is interchangeable, implemented as plugins, primary as custom MessageBus or SMTP. The GUI is written as a WPF desktop application on top of PRISM framework and DI Unity. The first automation layer reduces duplicate or similar alerts with grouping or loss compression. A machine learning based classifier separates alerts for escalation. Second automation layer is built on top of rule-system with a custom grammar, templating engine and integration with external systems. It generates instructions how to solve the problem, search contacts and escalate to responsible persons. The system can be adjusted to align with actual business processes, external systems and network infrastructure. It provides very efficient cost-saving solution with increase of quality of provided services.

Key Words: Antivirus monitoring, Automation, Machine Learning, WPF, WCF, PRISM, Unity, Dependency Injection, MEF, Windows Service, ANTLR, C#, R

Obsah

Seznam použitých zkratek a symbolů

AES	–	Advanced Encryption Standard
ANTLR	–	ANother Tool for Language Recognition (název produktu)
API	–	Application programming interface
AV	–	Antivirus
BT	–	Gradient Boosted Trees
CMDB	–	Configuration management database
CRUD	–	Create, read, update and delete
DI	–	Dependency Injection
DMZ	–	Demilitarized zone
DT	–	Decision Tree
EAI	–	Enterprise Application Integration
EPP	–	Endpoint Protection
ETL	–	Extract-Transform-Load
GUID	–	Globally Unique Identifier
GUI	–	Graphical User Interface
IPS	–	Intrusion Prevention System
ITIL	–	Information Technology Infrastructure Library
ITSM	–	IT Service Management system
MDI	–	Multiple Document Interface
MEF	–	Managed Extensibility Framework
ML	–	Machine Learning
MVVM	–	Model-View-ViewModel
NN	–	Neural Networks
PRISM	–	Patterns and practices WPF composite application framework
RF	–	Random Forest
SIEM	–	Security Information and Event Management
SLA	–	Service Level Agreement
VPN	–	Virtual Private Network
WCF	–	Windows Communication Foundation
WPF	–	Windows Presentation Foundation
XAML	–	Extensible Application Markup Language

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Práce se zabývá návrhem a implementací systému pro monitorování antivirových programů spravovaných firmou Tieto s podporou strojového učení a jiných technik k automatizaci procesů. Navržený systém je rozšiřitelný o další problémové domény jako je IPS, SIEM, Vulnerability Management aj.

V úvodní kapitole je čtenář seznámen s činností týmu, pro který bude výsledný systém navržen, a také s klíčovými pojmy z řešené oblasti. Následně jsou prozkoumány existující řešení a jejich vlastnosti.

V další kapitole jsou analyzovány požadavky na tento systém a je navržena jeho rámcová architektura. Následuje konkrétní návrh a implementace Windows služeb pro přenos dat a příkazů spolu s grafickým rozhraním pro konfiguraci systému a centrální monitorování alertů formou desktopové GUI aplikace.

Od kapitoly 6 se práce věnuje automatizačním úlohám. Jsou představeny procesy a identifikovány části vhodné k automatizaci. U procesů řešitelných strojovým učením jsou vybrány a teoreticky popsány vhodné algoritmy. Pro další automatizační úlohy je navržena vlastní gramatika, která slouží jako základ pro pravidlové systémy. Ty řeší sestavení vhodných instrukcí k danému problému a eskalaci na zodpovědné osoby nebo týmy. Poslední automatizační úloha řeší koncept nespojového volání API na vzdálených systémech z centrálního monitoringu a zpracování jejich odpovědi.

Poslední kapitola se zabývá tvorbou modelu z monitorovaných antivirových alertů, nad kterým jsou následně otestovány vybrané algoritmy strojového učení na přesnost a výkon. Na základě těchto testů je sestaven finální klasifikátor pro automatizaci pomocí strojového učení.

1.1 Firma Tieto

Tieto je největší IT společností v severní Evropě, která poskytuje outsourcing IT služeb a vývoj firemních systémů zákazníkům s globálním rozsahem podnikání. Počátky sahají do roku 1968 v Helsinkách, kdy firma poskytovala služby převážně pro Union Bank of Finland a její zákazníky, spolu s několika firmami z oblasti lesnického průmyslu. V průběhu uplynulých let, spolu s rozvojem IT, se rozrostla do současného stavu, kdy zaměstnává přes 13 000 expertů z téměř dvaceti zemích světa s obratem přibližně 1,5 miliard €.

1.2 Tieto Security tým

Bezpečnostní tým firmy Tieto poskytuje outsourcing a konzultace ICT Security služeb mnoha zahraničním společnostem z různých zemí, které se liší nejen rozsahem svých sítí (jednotky až desítky tisíc klientů), ale také zaměřením a požadavky na bezpečnost, od státních organizací (orientace na cenu), přes průmysl až po bankovní sektor (nejvyšší bezpečnostní požadavky).

Současný model outsourcingu ICT služeb je založen na frameworku ITIL[1] v3, který určuje rámcovou strukturu, globální procesy a role jednotlivých vrstev technické podpory. Ty jsou za posledních deset let téměř neměnné, i když neustále vycházejí aktualizace tohoto frameworku. V současnosti je také velmi pravděpodobný přechod na jiný model, který bude lépe odpovídat potřebám firmy Tieto, což bude zahrnuto v počáteční analýze požadavků.

Naopak ICT Security je v současné době jedním z nejrychleji rostoucích oborů a tak se interní procesy a portfolio služeb dynamicky velmi rychle mění. Pokud v nepříliš dávné minulosti (rok 2004, kdy vznikla divize Tieto Czech) představovalo standardní zabezpečení stanic antivir, který rozpoznával cca 30 000 virů, bez IPS[2], častokrát i bez firewallu, pak při rozvoji internetových služeb, narůstajícím množstvím uživatelů, typů útoků, hodnotě digitalizovaných dat, frameworků pro tvorbu malware a dalších vlivů, se nacházíme ve stavu, kdy je v roce 2016 přes 600 milionů druhů malware[3] a denně vzniká dalších 390 tisíc nových mutací či rodin (viz [1]).

[1] ITIL (Information Technology Infrastructure Library) je framework pro řízení ICT služeb na základě best practices, vydaných jako sada pěti knih: Service Strategy, Service Design, Service Transition, Service Operation a Continual Service Improvement. Více na `https://www.axelos.com/best-practice-solutions/itil`

[3] Malware je obecnější název pro počítačové viry, který zahrnuje veškeré typy škodlivých programů (Ransomware, Backdoor, Spyware, Trojan atd.), tj. těch které nepotřebují ke svému šíření hostitele.

[3] Intrusion Prevention System skenuje síťovou aktivitu na známé typy útoků, kde díky svým signaturám umí rozpoznat a zabránit známým typům útoku. Variantou pro klienty je pak HIPS (Host IPS), který sleduje i aktivitu na úrovni spouštěných procesů.

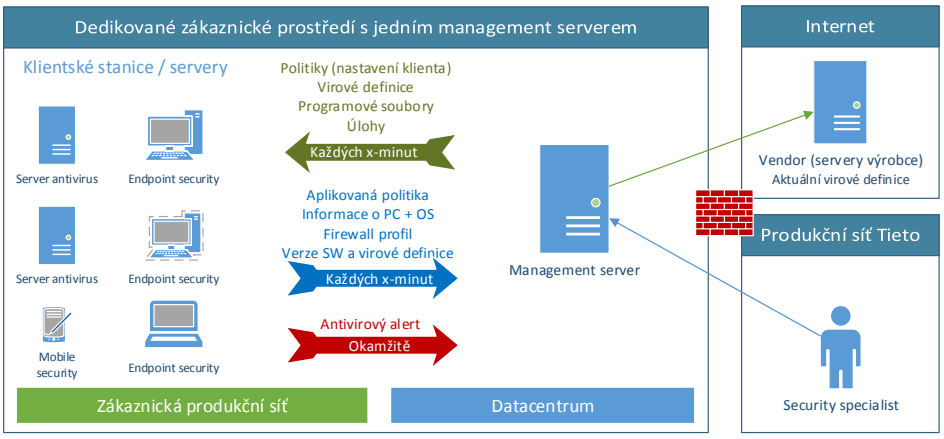
1.3 Monitoring antivirů a dalších prvků bezpečnosti

V následujících kapitolách se seznámíme se základními principy a pojmy ze správy antivirů.

1.3.1 Management server

Každý antivirový[4]systém v podnikovém prostředí podporuje centrální správu. Ta je zajištěna jedním či více servery (dále **Management Server**), na které se jednotliví klienti připojují, získávají svá nastavení (dále **politiky**), aktualizují své virové databáze nebo programové soubory, reportují svůj aktuální stav (použitá virová definice, údaje o počítači a OS, síti a další atributy, dle možností konkrétního produktu) a také na své Management servery nahrávají všechny události, na které jsou nakonfigurovány.

Tieto provádí outsourcing těchto systémů ve dvou modelech, dedikované servery, kdy je celé prostředí včetně management serverů umístěno v síti zákazníka a pronájem, kdy jeden zákazník sdílí management server s dalšími zákazníky, což je typické pro cloudové služby nebo jako ochrana serverů v DMZ a jiných lokací, ze kterých není žádoucí mít přístup na zákaznický management server, byť se jedná o zákaznické servery.



Obrázek 1: Zjednodušené schéma dedikovaného AV prostředí jednoho zákazníka

K management serverům je možné přistoupit přes VPN do zabezpečené mezi-sítě a následně přes Jumpoint servery, které slouží jako prostředník mezi Tieto a zákaznickou sítí. Stejným způsobem přistupujeme ke sdíleným serverům, které jsou také umístěny mimo produkční síť, a je nutné přistupovat přes vícevrstvou ochranu. Shrnutí: žádný management server není možné přímo propojit s Tieto produkční sítí v žádném směru, viz kapitola 4.

[4]Pro jednoduchost budeme používat termín Antivirový systém, přestože se v dnešní době obvykle používá tzv. Endpoint Security, které zahrnuje sadu modulů pro ochranu koncových stanic, zpravidla se jedná o Antivir, Firewall a IPS (analogicky k tomu existuje i Server Security). Dodatečně mohou obsahovat další prvky ochrany jako je zabezpečené prohlížení internetu, komunikace s bankou, rodičovská ochrana apod.

1.3.2 Antivirový alert

V okamžiku, kdy antivir detekuje podezřelý soubor a provede předem definovanou akci (dle nastavení - smazat, pouze reportovat, karanténa, vyčistit...), je na management server nahrána událost, kterou budeme označovat jako **antivirový alert**. Ten obsahuje minimálně informace o detekovaném souboru nebo procesu, signatuře nebo metodě[5], kterou byl malware detekován, verzi virové definice, času a provedené nebo předpokládané akci[6]. Obvykle obsahuje další informace, jako jsou údaje o uživateli, počítači a operačním systému, v případě napadení ze sítě také údaje o útočícím počítači. Detailní struktura obecného antivirového alertu bude rozebrána v samostatné kapitole.

Antivirový alert reprezentuje bezpečnostní riziko, které je zhodnoceno analytikem a vyžaduje adekvátní reakci. Dle povahy a bezpečnostních politik zákazníka může zareagovat čtyřmi způsoby, pokud budeme vycházet z obecně platných způsobů jak nakládat s bezpečnostními riziky[7] (viz [2, strana 250]):

- *Reduce or mitigate* - provede akce ke snížení rizika, např. preventivní blokace souboru/hashe na úrovni globálních politik.
- *Assign or transfer* - představuje základní reakci na stav, kdy antivir detekoval infekci, ovšem nebyl schopen zabránit její další činnosti (nebo byl nastaven pouze pro detekci) nebo opětovné re-infekci[8]. Analytik sepíše instrukce k odstranění malware a předá je na zodpovědný tým či osobu (podle konkrétního kontraktu na service desk, server tým, bezpečnostní oddělení zákazníka apod.). Dále budeme tuto činnost nazývat **eskalace**.
- *Accept* - může být např. použito v případě, kdy zákazník vědomě používá potencionálně nebezpečný software, nebo je napaden malware, kde náklady na odstranění přesahují škodu, kterou z pohledu globální bezpečnosti může daný malware napáchat (např. Tracking cookie), nebo se jedná o false positive, na který již byly provedeny preventivní akce na základě dřívějšího alertu (problém byl eskalován na výrobce, ale chybná detekce ještě nebyla odstraněna z aktuálních definic), testovací alert apod.
- *Reject or ignore* - aneb nedělat nic je možná varianta reakce, ale ne přípustná jako součást nabízených služeb.

[5] Heuristická detekce, cloud reputation a jiné techniky jsou v dnešní době nedílnou součástí antiviru, kdy doplňují klasickou detekci na bázi virových signatur o schopnost zachytit zero-day malware.

[6] Pokud není možné soubor smazat, může být akce naplánována po restartu počítače.

[7] Bezpečnostní riziko se nedá eliminovat, dá se pouze snížit na akceptovatelnou hranici, převést zodpovědnost (nebo se proti takové škodě pojistit), riziko akceptovat či předpokládat že nemůže nikdy nastat.

[8] Obvykle dvojice malware, např. nedetekovaný Trojan.Downloader, který stahuje Spyware, jenž umí antivir detekovat. Dojde k situaci, kdy antivir smaže stažený malware, ale neumí detekovat Downloader, který infiltroval počítač, čímž dochází k neustálé reinfikaci počítače a generování nových alertů.

1.3.3 Outsourcing antivirových služeb

Tieto podporuje antivirové systémy od firem Intel (McAfee), Symantec a F-Secure. Konkrétní služby a jejich SLA[9]jsou dány servisní smlouvou s každým zákazníkem zvlášť, vycházejí však ze standardního, katalogového "Service Description", obecně je lze rozdělit na:

- Správa management serverů a prostředí - nastavení politik dle rolí klientů, údržba aktuálních virových repositářů pro klienty, zálohování, přechod na vyšší verze, tvorba instalačních balíčků aj.
- Podpora uživatelů a řešení provozních problémů.
- Monitoring antivirových alertů - sledování a reakce na bezpečnostní incidenty inicializovanými antivirovými alerty.
- Instalace a vyřazení z provozu serverových antivirů (severy jsou vázané na další služby, z tohoto důvodu jsou tyto dvě činnosti součástí change procesu, viz ITIL)).
- Reporting - periodický na měsíční či týdenní bázi, ad-hoc a interní reporty.

V současné době má Tieto Security tým pod správou cca **200 tisíc monitorovaných klientů**, jejichž antiviry vygenerovaly **v roce 2016** přibližně **2,5 milionů alertů**. Tento trend má každoročně vzrůstající charakteristiku, což je nepříznivé z pohledu potřeby růstu každé komerční společnosti, neboť by udržení stávající kvality spolu se získáním nových zákazníku vyžadovalo neúměrné rozšiřování týmu.

Antivirový program generuje nový alert v poměru 1:1 na každý detekovaný soubor zvlášť. Bezpečnostnímu analytikovi tak přichází z jednoho infikovaného počítače množství redundantních dat (extrém za rok 2016 činí 210 000 alertů z jednoho počítače v jediný den), ze kterých pak získává velmi málo nových informací, častokrát nepotřebných pro správné rozhodování.

Z celkových 2,5 milionu alertů vyžadovalo eskalaci jen 640 tisíc, dalších 54 tisíc byly false positive[10]. Tyto hodnoty však neukazují realitu, v praxi se celý problém redukuje na tzv. incident, který představuje počítač infikovaný určitým malware se seznamem infikovaných souborů. Tímto způsobem se dá množství alertů zredukovat na 300 tisíc incidentů, které vyžadovaly vytvoření 18 000 eskalací zaslaných přes ITSM (IT Service Management systém, např. ServiceNow, BMC Remedy, Jira) anebo email zodpovědnému týmu či osobě.

[9]SLA - Service Level Agreement, definuje služby a hraniční časy, při jejichž překročení jsou účtovány sankce. Obvykle se jedná o dobu, do kdy musí být vyřešen uživatelský požadavek na podporu, maximální prodleva aktualizace definice od jejího vydání, čas reakce na antivirový alert aj.

[10]False positive označujeme chybně detekovaný soubor, který neobsahuje žádnou infekci a je způsoben chybou ve virové definici. Například antivirový alert na součást Windows nebo Office jako je MS Word, v praxi je převážná většina false positive generována zákaznickými aplikacemi. U některých agresivních heuristických modulů antivirů stačí vytvořit aplikaci v C, která obsahuje jediný řádek, otevření souboru pro čtení a tento program bude smazán ihned po kompilaci. False positive je eskalován na výrobce k opravě definic.

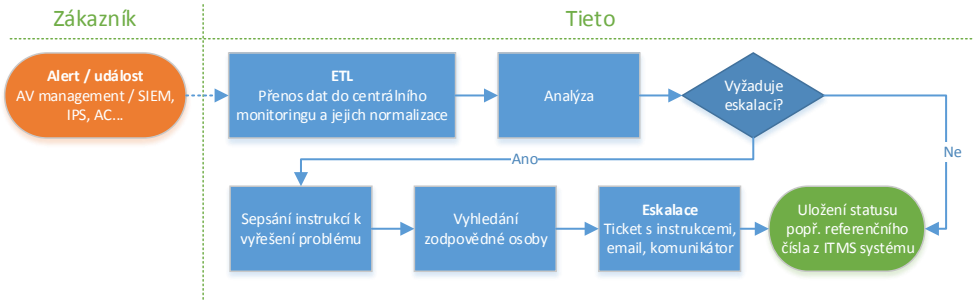
1.3.4 Další monitorované systémy

Samotné antiviry by k zabezpečení sítě v roce 2017 samozřejmě nestačily, bezpečnostní oddělení musí poskytovat i další služby jako je Intrusion Prevention System (IPS), Security Information and Event Management (SIEM), Next Generation Firewall (NGFW), Data Loss Prevention (DLP), Application Control (AC), Vulnerability Management (VM), Security Early Warning (SEW - sledování nových zranitelností, které afektují zákazníka), Security Proxy aj., které vyžadují monitoring a analytickou činnost.

Z tohoto důvodu je nutné co nejvíce zredukovat nadbytečnou práci, kterou je možné zautomatizovat a umožnit analytikovi, aby se mohl více zaměřit na řešení problémů a korelaci událostí mezi jednotlivými systémy.

1.3.5 Alert flow - klíčové procesy k automatizaci

Na obrázku 2 je znázorněn zjednodušený high-level diagram, který zachycuje základní úlohy při monitorování alertů a je obecně platný i pro další systémy, které poskytujeme. Jejich princip činnosti je velmi podobný, s výjimkou SIEM, který je možné považovat za mezivrstvu, neboť se jedná o systém pro shromažďování a analýzu logů obecně z jakéhokoliv systému, kde pomocí korelačních pravidel můžeme nahradit velkou část analytické práce. I ten však vyžaduje následnou manuální činnost, kterou je možné automatizovat. Navíc vzhledem k ceně si jej nemohou dovolit všichni zákazníci. Tato práce bude řešit antivirové služby, ale očekává se vytvoření systému, který bude moci s minimálním úsilím rozšířit o další služby.



Obrázek 2: Zjednodušený proces monitorování alertů

Pokud se podíváme na zjednodušený proces (obr. 2), pak vidíme, že každý alert vyžaduje analýzu, v případě eskalace sepsání potřebných instrukcí a dohledání, komu je zaslat. Tyto tři kroky představují velmi velkou pracovní zátěž, přitom se jedná z větší části o opakovanou činnost, tudíž takovou, která nabízí velký prostor automatizaci. S využitím technik strojového učení, vytvořením vhodných pravidlových systémů a automatizací podpůrných činností lze očekávat výrazné snížení pracovní zátěže a zároveň zvýšení kvality poskytovaných služeb. Více v samostatné kapitole věnované analýze požadavků (kapitola 4).

2 Cíle práce

Cílem práce je navrhnout modulární systém pro centralizovaný monitoring alertů z oddělených zákaznických prostředí spravovaných firmou Tieto a implementovat v něm modul pro monitorování antivirových alertů, který je schopen automatizovaného rozhodování na základě získaných zkušeností a aktivního snižování režie potřebné k reakci na incident operátorem. Systém je nutné navrhnout tak, aby jej bylo možné rozšířit o další vstupy pokrývající bezpečnost IT infrastruktury (IPS, SIEM, Vulnerability management...), podporu IT procesů v Security týmu, moduly podporující / monitorující automatizační úlohy v cloudovém řešení a další. Systém musí zohledňovat aktuální trendy DevOps+ITIL modelu správy IT, kladoucí maximální důraz na automatizaci, rychlost a počet změn uváděných do praxe, transformaci okolních služeb, integraci dat, auditování, spolupráci na vývoji systému v rámci týmu s různým stupněm znalostí a zaměření. Dílčí cíle práce jsou:

- Návrh architektury a implementace systému (vychází z původní bakalářské práce, věnované kompozitním aplikacím a dále na současné kolekci aplikací, které využíváme pro podporu stávajících služeb, jejichž jsem autorem):
  - Systém přenosu dat - události, statusy klientů (assets) a příkazů, který je snadno rozšiřitelný o nové zdroje s maximální znovupoužitelností existujících komponent a nezávislostí na přenosové vrstvě.
  - Systém pro strojové učení, který bude přes navržené rozhraní umožňovat klasifikovat výstup na základě naučeného modelu.
  - Aplikace s GUI pro administrátory a koncové uživatele, umožňující vizuální monitoring, interaktivní reakci na události, správu a nastavení služeb, zákazníků, uživatelů a oprávnění aj., kde stejným postupem pro vytvoření pluginů bude možné systém rozšířit o zcela nové služby, podpůrné nástroje, reporting a další moduly vhodné k automatizaci procesů.
  - Server pro hostování podpůrných síťových služeb a automatického spouštění naplánovaných událostí, které budou naprogramovány jako pluginy pro tento server.
- Výběr a testování vhodného algoritmu strojového učení, který bude schopen redukovat práci při zpracování alertů.
- Podpora automatizovaného sestavení instrukcí vhodných k řešení daného problému.
- Automatizace vyhledávání zodpovědných osob, kontraktů a eskalace do externího systému.
- Spouštění vzdálených příkazů z centrálního monitoringu a aplikace odezvy z těchto systémů.

3 Průzkum existujících řešení a jejich vlastností

V současné době existuje na trhu několik produktů, věnující se dílčím částem řešených problémů, určených pro bezpečnostní analytiky a Security Operations Center (SOC).

Převážná většina těchto produktů má jako základ SIEM[11], nad jehož výstupem jsou pak napsány různé vizualizační nebo integrační aplikace.

SIEM je schopen vyřešit parsování logů, jejich uchování, korelační engine umožní filtrování duplicit, sestavení inteligentních pravidel pro generování alarmů atd. SIEM je modulární systém, jehož základ tvoří dlouhodobé uchování logů, nevýhodou je bohužel vysoká cena (daná schopností uchovat velké množství dat), která roste s každým dalším modulem. V základu umí pouze uchovávat logy a dle nastavených kritérií z nich generovat alarmy. Pokud se nejedná o standardní vstupní formát (Windows, Syslog aj.), tak si parsery musíme napsat sami nebo využít placené služby výrobce. S narůstající cenou rostou jeho schopnosti a to včetně podpory algoritmů pro strojové učení, v rámci correlation engine modulu.

Pokud zákazník vlastní SIEM nebo jej má v pronájmu, tak to přináší řadu výhod, které využíváme ke zpracování dat z jiných produktů, charakteristických velkým objemem událostí. Například všichni zákazníci, kteří se pohybují v oblasti platebních karet, toto zařízení musejí vlastnit nebo si jej pronajímat, jinak by nesplnili normu PCI DSS[12]. V této práci je však nutné vyřešit i zákazníky, kteří SIEM nemají a být schopni jim nabídnout antivirové služby s nulovými náklady na dodatečné systémy. Krom toho SIEM neřeší jeden ze základních požadavků a tím je Asset Management, takže by přenos informací o spravovaných klientech musela řešit extra služba a samozřejmě se nezabývá podporou našich interních procesů a využitím API antivirových managementů nebo řízením přes přímý přístup do databáze.

I když je SIEM schopen aplikovat chytré pravidla pro redukci nepotřebných dat a vytvářet hodnotnější alerty, tak automatizuje jen určitou část tohoto procesu. Pokud se vrátíme k úvodnímu příkladu, pak bychom pomoci SIEM mohli snížit počet vstupních alertů, za cenu že všechny pravidla budeme muset spravovat na každém SIEM zvlášť, navíc nám chybí moduly pro eskalaci těchto alarmů a jejich instrukcí.

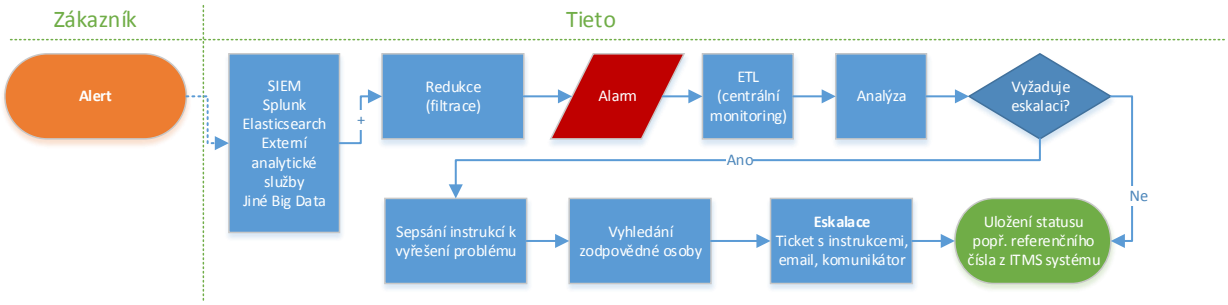
[11] SIEM- Security Information and Event Management, je systém pro správu bezpečnostních informací a událostí, umožňující dlouhodobé ukládání logů (pro potřeby PCI/DSS je povinný jeden rok) a také pro monitorování v reálném čase, kdy je schopen na základě korelačních pravidel generovat z daných logů alerty včetně filtrace duplicit aj.

[12] PCI DSS - Payment Card Industry Data Security Standard je mezinárodně platný soubor pravidel, které musí dodržovat společnosti, nakládající s platebními kartami. Ve zkratce definuje soubor pravidelných kontrol, které musí společnost splňovat, aby tento standard dodržela. Kontroly probíhají v pravidelných intervalech a jejich výstupem je tzv. PCI DSS compliance, tj. splňuje / nesplňuje. Jedním z mnoha vstupů je i SIEM, který umožňuje generovat realtime alarmy při zjištění, že by daný zákazník tuto normu porušil a umožňuje tak včasnou opravu. Více na oficiálních stránkách, viz [3].

(Pozn.: zpracování alarmů ze SIEM je jedním z plánovaných modulů navrhovaného systému, protože SIEM neřeší sepsání instrukcí, eskalaci a follow-up eskalačního procesu na alarmy, generované dle korelačních pravidel z nashromážděných logů.)

Téměř každé komerční SIEM řešení nabízí nadstavby zaměřené na Security. Namátkou Intel / McAfee Enterprise Security Manager, IBM Security QRadar SIEM, Splunk Enterprise Security a další. Tyto produkty nabízejí řadu zajímavých funkcí včetně forenzní analýzy, Threat intelligence, některé také integrace s ITSM. Krom uchování logů nabízejí navíc častokrát dobrou vizualizaci dat a jsou velmi užitečné pro hledání problémů. Ovšem i po všech korelačních pravidlech, filtraci duplicit a dalších akcích je výstupem alarm bez řešení a navržení správné zodpovědné osoby / týmu. Stejně jako u základních SIEM produktů je potřeba počítat s vyšší cenou, kdy platíme jak za samotný SIEM, tak za jeho security nadstavbu. Zajímavou open-source platformou je Elasticsearch stack, který ve spojení modulů Elasticsearch+Kibana+Logsash+Beats nabízí kompletní řešení pro sběr, přenos, vizualizaci a analýzu logů, nebo placené Microsoft Azure - Operations Management Suite.

SIEM a platformy založené na Big Data, jsou pro navrhované řešení vhodné k před-zpracování objemných dat z firewallu, DNS, audit logů aj. nebo také pro ruční analýzu původních dat, které by bylo nákladné uchovávat v centrálním monitoringu a je vhodné je předřadit jako zdroj alarmů, generovaných na základě korelačních pravidel nebo je využít pro vizualizaci a detekci anomálií.



Obrázek 3: Proces doplněný o předzpracování na Big Data platformě

Na trhu se občas objeví zajímavý open-source projekt, bohužel však nemívá dlouhodobější životnost, jako např. Big Data řešení OpenSOC[13] nebo FIDO[14], který formou kostry svého interního nástroje pro automatickou odezvu s integrací do firemní infrastruktury uvolnila firma Netflix.

V rámci průzkumu existujících řešení nebyl nalezen systém, který by komplexně pokryl potřeby našeho týmu, ale byly prozkoumány systémy, které vhodně doplňují celkové řešení.

[13] OpenSOC - `http://opensoc.github.io/`
[14] Netflix/FIDO - `https://github.com/Netflix/Fido`

4 Analýza požadavků a návrh systémů

Základní požadavky na systém vycházejí z cílů práce (viz kapitola 2), nyní se pokusíme upřesnit jednotlivé části a navrhnout rámcovou architekturu systému.

4.1 Bakalářská práce

V bakalářské práci, věnované vývoji kompozitních aplikací pomocí PRISM, byly představeny požadavky na architekturu software pro Security tým a vysvětleny rozdíly modulárního přístupu oproti klasickým monolitickým aplikacím. Hlavními požadavky byla možnost efektivně reagovat na změny v okolních systémech, tj. snadno přidávat a odebírat funkce, bez rizika zásahu do různých částí aplikace, znovupoužitelnost navržených komponent spolu s volitelnou možností izolovat práci na jednotlivých částech a vrstvách programu tak, abychom umožnili nezávislou práci více programátorů s možností psát unit testy i na uživatelské formuláře *(demonstrace potřeby návrhu a implementace na těchto principech byla provedena na snadno pochopitelném příkladu primitivní aplikace s uživatelským dialogem, do které byly postupem času implementovány nové a změnové požadavky)*.

V bakalářské práci byly představeny klíčové návrhové vzory[15]a pojmy, ze kterých tato diplomová práce vychází, způsob jakým je v PRISM spolu s DI kontejnerem řešeno sestavení aplikace z pluginů, vybrán Dependency Injection kontejner pro GUI aplikaci (Unity), navrženy základní aplikační služby, WPF komponenty a vysvětleny konkrétní techniky PRISM, podporující MVVM (separace prezentační, aplikační a datové vrstvy), dekompozice složitějšího GUI na regiony a navigace na View (formuláře) bez těsné vazby. Krom známých návrhových vzorů je tato problematika dobře zpracovaná v literatuře [4], [5] a [6].

V této práci budeme používat i Windows služby, u kterých bude obecně platit stejný požadavek. Služba bude obsahovat inicializační kód, který má za úkol načíst potřebné pluginy, plánovat a spouštět procesy přes předem definované rozhraní. Přidání a odebrání funkcí pak bude zásadně v režii jednotlivých pluginů.

DI kontejner pro interní služby bude odpovídat výběru z bakalářské práce, tudíž je shodný s kontejnerem pro GUI (byl vybrán Unity), což usnadní sdílení a přesun kódu z GUI na službu. Služby pro přenos dat ze zákaznických prostředí, budou založeny na MEF, který více odpovídá modelu pro rozšiřování typu 3rd party (podrobnější informace k MEF viz [7]).

[15]V práci jsou vysvětleny vzory Inversion of Control, Service Locator, Dependency injection, Sepeated presentation (Model-View-ViewModel (MVVM)), Command, Repository, Observer, Adapter, Composite a Composite View, Registry, Event Aggregator, Separated Interface, Plug-in, Façade, Application Controller.

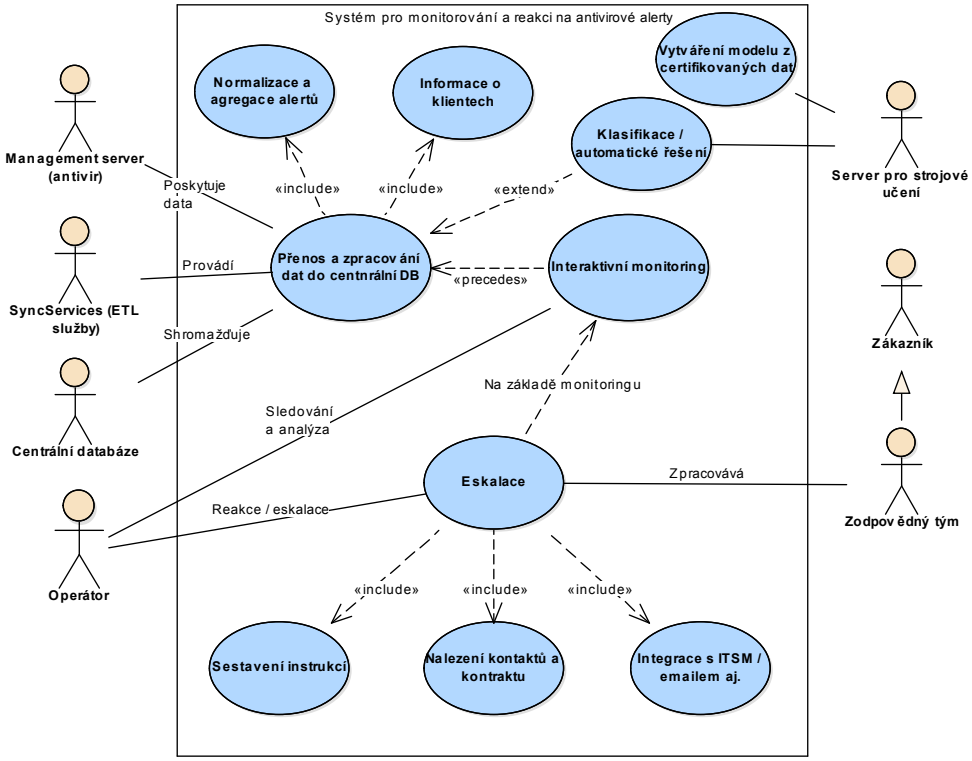
4.2 Současný stav

Aktuálně využíváme pro monitorování Windows služby a GUI aplikaci (autorské projekty), které nebyly navrženy pro současné potřeby, ale byly vyvinuty a následně rozvíjeny dle nabízených služeb v uplynulých deseti letech. Chybí zde klíčové koncepty, kterými by se jednotlivé části řídily a nejsou implementovány technologie ani filosofie návrhu zpracované v bakalářské práci. Její udržitelnost a náročnost implementace změnových požadavků je velmi vysoká, napojení na nové monitorované systémy zvyšuje nelineárně složitost celé služby a odebírání nepodporovaných systémů je také velmi obtížné.

Zároveň zde nejsou implementovány technologie z oblasti strojového učení, které dovedou efektivně nahradit opakovanou lidskou činnost, mnohdy s mnohem přesnějšími výsledky.

4.3 Požadavky na systém

Pomocí use case diagramu (viz obrázek 4) jsou identifikováni základní aktéři systému spolu s hlavním případem užití. Klíčové požadavky na systém si definujeme textově v následujících podkapitolách a rozvedeme k high level návrhu systémů.



Obrázek 4: Klíčové use case

4.3.1 Centralizace monitorovaných dat

Systém musí obsahovat mechanismus pro automatický přenos antivirových alertů ze zákaznických i sdílených prostředí do centrální databáze. Dále musí udržovat v centrální databázi informace o všech monitorovaných klientech a jejich aktuálním stavu (assets, v práci budeme využívat termín statusy klientů). Pro zákaznické prostředí, které umožňují obousměrný přenos dat, je nutné umět přenést příkazy z centrálního serveru na zákaznický management server a vrátit jejich výsledek. Systém musí vyřešit, jakým způsobem zabezpečeně přenést data přes nepřímo propojené sítě, ideálně tak, aby centrální databáze mohla zůstat v produkční síti Tieto a operátor by měl tím pádem volné VPN připojení (v opačném případě by byl nucen využívat virtuální počítač nebo vzdálenou plochu na server v datacentru a vyřešit otevření spojení mezi jednotlivými datacentry pro tento typ aplikace).

4.3.2 Interaktivní monitoring

Systém musí obsahovat uživatelské rozhraní, které umožní interaktivně eskalovat alerty operátorem, zobrazovat všechny dostupné informace obsažené v alertu, sledovat nezpracované i zpracované alerty dle nastaveného filtru (datum, produkt, zákazník, malware, akce a status). Operátor musí být schopen při rozhodování vycházet i z historických dat vztažených k infikované stanici či serveru a vidět předchozí eskalované alerty. V případě, že byl alert eskalován přes ITSM, pak by měl systém umožnit s minimálním úsilím vyhledat, jak byl daný tiket vyřešen.

4.3.3 Podpora sestavení instrukcí

Systém musí umět automatizovaně navrhnout a sestavit co nejvhodnější instrukce, podporující řešení daného problému, jejichž kvalita a detail bude úměrná úsilí potřebnému ke konfiguraci tohoto systému. Základní instrukce musí obsahovat informace z alertu, potřebné k identifikaci a řešení problému, postupy vztažené k afektovanému produktu a konkrétní situaci, musí používat identifikátory, adresy a zohledňovat procesy dle konkrétního zákazníka. Tento systém musí být rozšiřitelný o nové postupy, jako příklad může posloužit extenze, kdy při detekci na bázi antivirových signatur a použitém produktu může být využito externího zdroje dat nebo přidán odkaz na konkrétní řešení ze stránek výrobce antiviru aj.

Takto před-připravené instrukce bude mít operátor možnost upravit nebo doplnit dle potřeb při interaktivní reakci na konkrétní alert a stav zákaznického prostředí. Cílem tohoto požadavku je nejvyšší možnou měrou minimalizovat čas operátora pro sestavení instrukcí, eliminovat opakovanou práci a přitom jej navrhnout tak, aby nekladl vysoké nároky na čas strávený konfigurací šablon či podpůrných systémů, neboť pro jeho konfiguraci budou vyčleněni 2-3 administrátoři.

4.3.4 Eskalace

Podpora eskalace znamená, že systém sám připraví, komu a jakým způsobem budou instrukce doručeny, v případě ITSM také včetně všech povinných doplňujících informací. Následně se bude umět s daným systémem spojit a pomocí API eskalovat připravené instrukce k řešení, buď formou vytvoření ITSM ticketu, odesláním emailu nebo jiným způsobem, definovaným v budoucnosti. Zde je nutné zdůraznit, že většina monitorovaných systémů jsou koncové stanice, které bohužel nemívají záznamy a vazby v CMDB a tak relace na smlouvy a zodpovědné týmy musí řešit navrhovaný systém. V případě serverů je již standardem, že jsou v CMDB definovány kontrakty, zodpovědné týmy, afektované služby a další relace, takže zde je situace jednodušší.

Systém musí rozpoznat, z jakého zákaznického prostředí či její části alert pochází, který produkt daný malware detekoval a afektovanou platformu (tj. server / pracovní stanice). Dále musí podporovat nejméně tři úrovně eskalace a to standardní proces, reakce na false positive a eskalace na vyšší level v případě, že předchozí akce nebyly zpracovány korektně.

Na základě těchto dat musí vyhledat, který tým je za tento počítač a službu zodpovědný, připravit údaje pro vytvoření eskalace na konkrétní ITSM (nalezení kontaktů, kontraktu, zodpovědného týmu) nebo pro jiný komunikační kanál, jako je email, nalézt adresu příjemce a odesílatele. Metody pro nalezení zodpovědného týmu či osoby mohou být v budoucnosti rozšířeny. Cílem je eliminovat sdílené podpůrné dokumenty v kancelářských programech (typicky sada Word či Excel dokumentů). Systém musí vyhledávat jak ve vlastní databázi, tak v podporovaných externích zdrojích dat.

Pro pochopení tohoto požadavku si můžeme představit firmu, skládající se z několika oddělení, které fungují nezávisle nebo firmu, která má pobočky ve více státech. Jejich koncové počítače mohou mít pod správou jejich vlastní IT oddělení nebo lokální podporu pro jednotlivé země anebo externí firmu pro každé město zvlášť, zatímco servery jsou kompletně spravovány firmou Tieto v jejich data-centrech, případně si je zákazníci spravují sami. V praxi existují všechny představitelné kombinace. Účetní oddělení může mít definovány jiné kontrakty, SLA a zodpovědné týmy než oddělení pro podporu zákazníků. Stejně tak globální korporace, které mají pobočky v mnoha státech světa s vlastními týmy pro lokální podporu, budou vyžadovat, aby pro počítač umístěný v Kanadě byl kontaktován Kanadský service desk nebo přímo tým lokální podpory v daném městě. Informace mohou být vyhledány pomocí předdefinované konfigurace, CMDB, Active Directory, externí databáze, Lotus Notes a jiných, zákazníkem používaných systémech.

4.3.5 Zabezpečení

Přístup k jednotlivým modulům a jejich funkcím, musí být zpřístupněn na základě uživatelských práv, definovaných na určitou roli v systému. Uživatel může být členem žádné až všech dostupných rolí, vítězí nejvyšší oprávnění z přiřazených rolí. Systém by měl umožnit nastavení práv pro teoreticky libovolný prvek, jak formuláře a ovládací prvky, tak samotná data, přičemž všechny funkce z problémových domén jsou řešeny formou pluginu, kterými se rozšiřuje základní GUI.

4.3.6 Strojové učení

Systém musí umět aktivně využívat znalostí získaných z rozhodnutí provedených operátorem k minimalizaci práce operátora při řešení stejného či podobného problému. Všechny nerozhodnutelné situace by měly být ponechány na operátorovi, cílem využití strojového učení je maximální úspora času při minimálním riziku chybného rozhodnutí a tak v případě překročení toleranční hranice by mohl systém maximálně navrhnout vhodné řešení, než aby došlo k nesprávnému rozhodnutí. Důvod tohoto chování je velmi prostý, zatímco náklady na monitoring se mohou pohybovat v řádu jednotek až desítek tisíc eur za měsíc, pak výpadek zákaznického prostředí může znamenat desítky tisíc až milióny eur za jediný den.

Stejně tak musí systém reagovat na výrazné výkyvy v celkovém prostředí, tzv. outbreak situace, kdy je velká část počítačů jednoho zákazníka napadena rychle se šířícím malware nebo je afektovaná false positive detekcí a přejít na outbreak proces, který není součástí standardního monitoringu a vyžaduje urgentní řešení na míru.

4.4 Monitorované systémy

Každý výrobce vyvíjí zpravidla několik bezpečnostních produktů, cílených na jiné zařízení a uživatele, které jsou však bez výjimky řízené jedním management serverem daného výrobce. V praxi tedy platí, že nejnovější verze antivirového management serveru daného výrobce umí spravovat všechny jeho podporované antivirové produkty. Systém musí v základu podporovat McAfee ePolicy Orchestrator (ePO), F-Secure Policy Management Server (FSPMS) a Symantec Endpoint Protection Management Server (SEPM).

S každou generací management serveru nebo změnou hlavní verze (dle konceptu výrobce), se ve většině případů změní i interní struktura databáze nebo API rozhraní. Systém musí být navržen tak, aby umožnil snadné přidávání podpory nových verzí nebo produktů jiných výrobců.

4.5 Antivirový alert, popis

Počet a názvy atributů se liší dle použitého produktu. Standardní antivir umí informovat o základních datech v rámci lokálního počítače (název počítače, soubor, malware, akce, uživatel, čas, operační systém). Oproti tomu pak Endpoint Security informuje i o případném zdrojovém počítači a jeho IP adrese, URL adrese stránek, ze kterých byl malware stažený, název aplikace a certifikát výrobce, proces ze kterého byl spuštěn a další. Terminologie pro názvy atributů se liší dle jednotlivých výrobců a proto je nutné tyto názvy znormalizovat, viz výpis 1.

```
Alert date: 24.04.2015 06:32:22
Customer name: Tieto
Computer WINS name: TESTCOMPUTER
Computer DNS name: testcomputer.sample.com
User account: SYSTEM
Source IP address: 10.100.32.131
Group path: \My Company\Workstations\Czech − CZ
Computer UID: C676CBE12C80088C016ED6F0AE785000
OS name: Microsoft Windows
OS version: 8.1
Platform type: Workstation
Severity: Major
Product vendor: Symantec
Product version: 12.1.4112.4156
Signature version: 2015−04−23 rev. 020
Scan type: On Access Scan
Infection: SecurityRisk.OrphanInf
Application: autorun.inf
Filename: e:\autorun.inf
Category: File virus
Description: Category: Security Risk / Location: File virus / Detection: Heurictic
Action: Quarantined
```

Výpis 1: Zobecněný antivirový alert

Model pro strojové učení, založený na původních hodnotách by nebyl příliš vhodný, proto je nutné z původního alertu vypočítat kategoriální atributy, viz výpis 2. Důvod je zřejmý z tabulky 1, která ilustruje množství a typ hodnot jednotlivých atributů. Reálně použité atributy a jejich normalizace budou známé až z výsledků zpracovaných v kapitole 7.1 a 7.2, již nyní však dovedeme odhadnout, jakým směrem by se mohla redukce ubírat.

```
CustomerSecurityAwareness: Low / Normal / High
PlatformType: Client / Server
MalwareType: Trojan, Worm, Ransomware, Virus...
RiskAssesment: Low / Medium / High / Critical
MalwareShortName: Exploit:Java/Majava.A, Exploit:Java/Majava.D => Exploit:Java/Majava
InfectedFolder: c:\Windows\Temp\file.exe => c:\Windows\Temp\
UserAccount: User / System
FileCategory: Executable / Autorun / Document / Installer / Archive / Temp...
FilePathCategory: ProgramFiles / ProgramData / Users / Temp / ShadowCopy / RecycleBin...
```

Výpis 2: Ilustrace normalizovaných hodnot z alertu, vhodných pro strojové učení

Pokud se podíváme na typy základních atributů a jejich zastoupení v již detekovaných alertech (viz tabulka 1) nebo jejich teoreticky přípustných hodnotách (vycházíme z údajů z roku 2016), pak je zřejmé, že některé atributy budou potřebovat značnou redukci dimenze, neboť by znamenaly obrovské množství kombinací, jenž by daný model musel obsahovat. Po zvážení počtu kombinací hlavních atributů a počtu lidských zdrojů, které je možné přiřadit pro sestavení a aktualizace modelu, je zřejmé, že budeme hledat algoritmus nebo skupinu algoritmů, u kterých se budeme muset rozhodnout, jestli mají být schopné se vypořádat s chybějícími hodnotami nebo s nimi bude počítat pracovní proces.

		Rozsah	
Atribut	**Typ**	Skutečný	Teoretický (2016)
Computer name	Kategoriální	150 000	150 000
Computer DNS name	Kategoriální	200 000	200 000
IP address	Kategoriální	185 000	200 000
Malware name	Kategoriální	55 000	600 000 000
Malware type	Kategoriální	10-50	10-50
OS name	Kategoriální	10-20	10-20
OS version	Kategoriální	50-100	50-100
Platform type	Kategoriální	2	2
Platform architecture	Kategoriální	2	2
InfectedFile	Kategoriální	700 000	N
Action	Kategoriální	7	7
Detection type	Kategoriální	2	2
Počet detekcí	Spojitá	1 - 432 000	N
Product	Kategoriální	20	20
Vendor	Kategoriální	4	4
Customer	Kategoriální	100+	100+

Tabulka 1: Počet hodnot dle atributů při modelování antivirového alertu

4.6 Návrh architektury systému

Na základě analýzy bude celý systém rozdělen na několik klíčových prvků. Jejich role a nasazení v jednotlivých sítích jsou znázorněny na obrázku 5.

4.6.1 Tieto.SyncServices.RemoteAgent

Windows služba, která má za úkol získat, transformovat a odeslat data na přenosové rozhraní nebo přijmout a spustit příkaz z centrálního serveru či naplánované úlohy (smazat server na vyžádání, provedení maintainance úlohy aj). Její nasazení je především v zákaznických prostředích, většinou přímo na management serveru.

4.6.2 Message Bus (Collector)

Cílem je implementovat vlastní nebo použít existující Message Bus s perzistentní frontou zpráv a strategií publisher-subscriber fungujícím v pasivním režimu, neboť bude umístěn v zabezpečené síti s povolenou pouze příchozí komunikací. Možné existující řešení (Kafka, NServiceBus, MassTransit aj.)

4.6.3 Tieto.SyncServices.Server

Windows služba, která má za úkol stáhnout datové balíky z přenosových rozhraní a naimportovat je do centrální databáze nebo odeslat příkaz službě Tieto.RemoteAgent viz 4.6.1.

4.6.4 Tieto.Security.Server (Job server)

Windows služba, která má za úkol spouštět naplánované úlohy (reporty, údržba, aktualizaci dat z interních systémů, synchronizaci CMDB CI aj.) a hostovat WCF služby pro funkce, které jsou inicializovány a GUI aplikace, ale spouštěny na serveru.

4.6.5 Tieto.Virusqueue

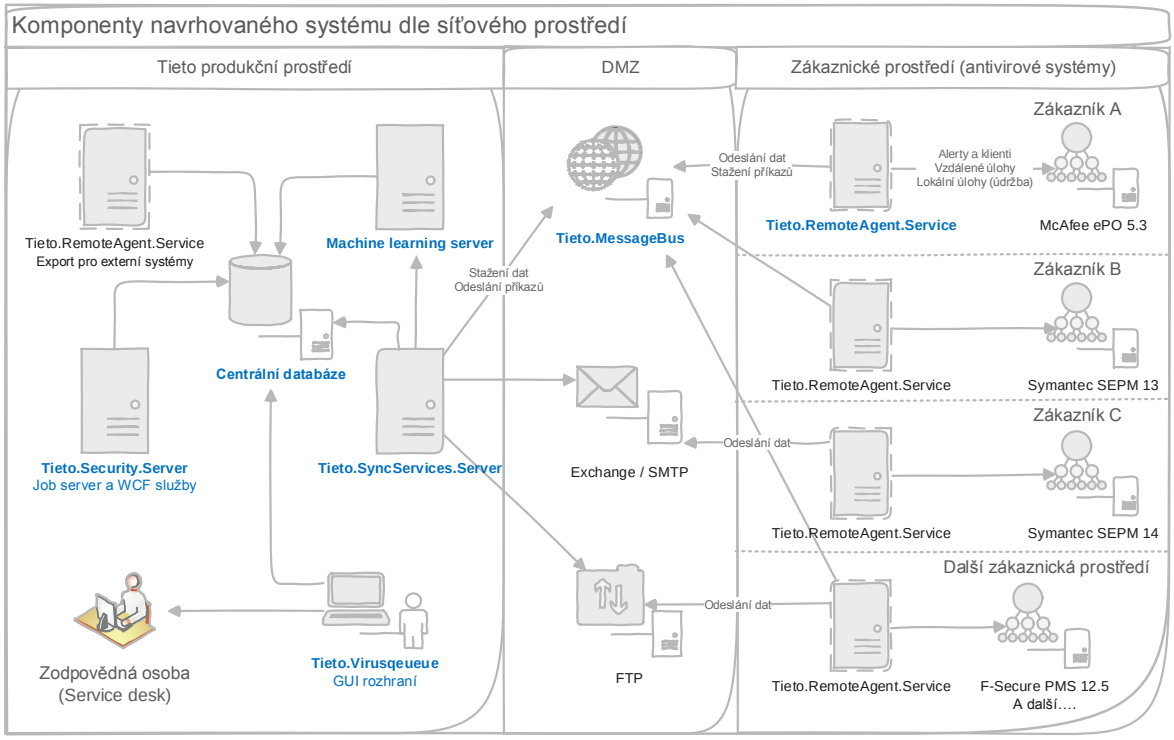
Grafická Windows aplikace s uživatelskými formuláři, které slouží pro monitorování alertů, konfiguraci systému, vyhledávání a práci s assety a podpůrné činnosti práce členů security týmu (utility).

4.6.6 Klasifikační služba

V předmětu MAD byla v rámci projektu testována klasifikace antivirových alertů s C# knihovnou Accord.NET a MS Analysis Services. Vzhledem k akvizici RevoScale firmou Microsoft máme možnost využít škálovatelných implementací ML algoritmů na platformě SQL Server R Services, která netrpí nedostatky zjištěnými v tomto projektu (paměťová náročnost, neprůhledný model aj.). Klasifikační služba bude realizována interně v rámci SQL Server uložených procedur nebo jako WCF služba.

4.6.7 Celkový přehled

Na obrázku 5 je pak znázorněno zjednodušené nasazení těchto komponent v zákaznickém a produkčním prostředí. (*pozn.: ve schématu nejsou znázorněné zřejmé detaily vyplývající z každé poskytované služby, jako je generování a doručení reportů a řada dalších nutných činností, o které se stará některá z uvedených komponent v produkční síti*).



Obrázek 5: Klíčové komponenty celkového systému

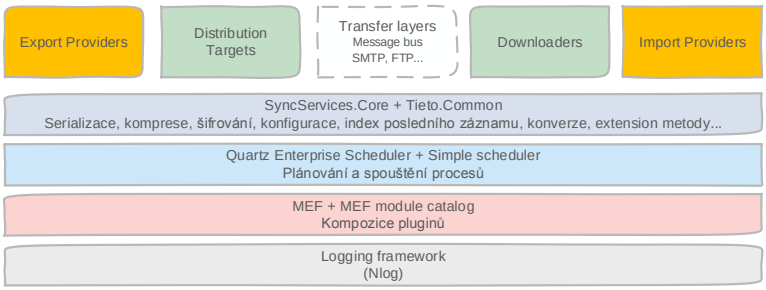
5 Návrh a implementace systému

V této kapitole si popíšeme detaily k návrhu a implementaci jednotlivých částí systému tak, jak byl na high-level úrovni navržen při analýze požadavků (kapitola 4.6).

5.1 Tieto.SyncServices

Tieto.SyncServices zajišťují primárně export a zabezpečený přenos dat ze vzdálených systémů v jednotném formátu do centrální databáze, tj. ETL proces[16], sekundárně pak dovedou zajistit spuštění příkazu z centrálního serveru na vzdáleném systému nebo spouštět pravidelné úlohy pro údržbu jako je odstranění duplicitních klientů, zálohování, aj., podle použitého modulu. Jejich nasazení je ilustrováno na schématu viz obr. 5.

V navrženém systému služeb jsou jednotlivé zodpovědnosti za každou část zpracování rozděleny tak, aby bylo dosaženo maximální znovupoužitelnosti jednotlivých komponent a zároveň jsme zajistili vysokou flexibilitu v oblasti nasazení a přenosu dat. Pokud se podíváme na obrázek 6, tak horizontální služby představují sdílený kód, který je společný pro všechny procesy, stejně jako kód, který zajišťuje práci s přenosovou vrstvou. Integraci nového systému do těchto služeb pak představuje napsání minimálního množství kódu izolovaného pouze na čtení dat z management serveru, v případě nové problémové domény pak vytvoření datové entity jejího uložení do centrální databáze.



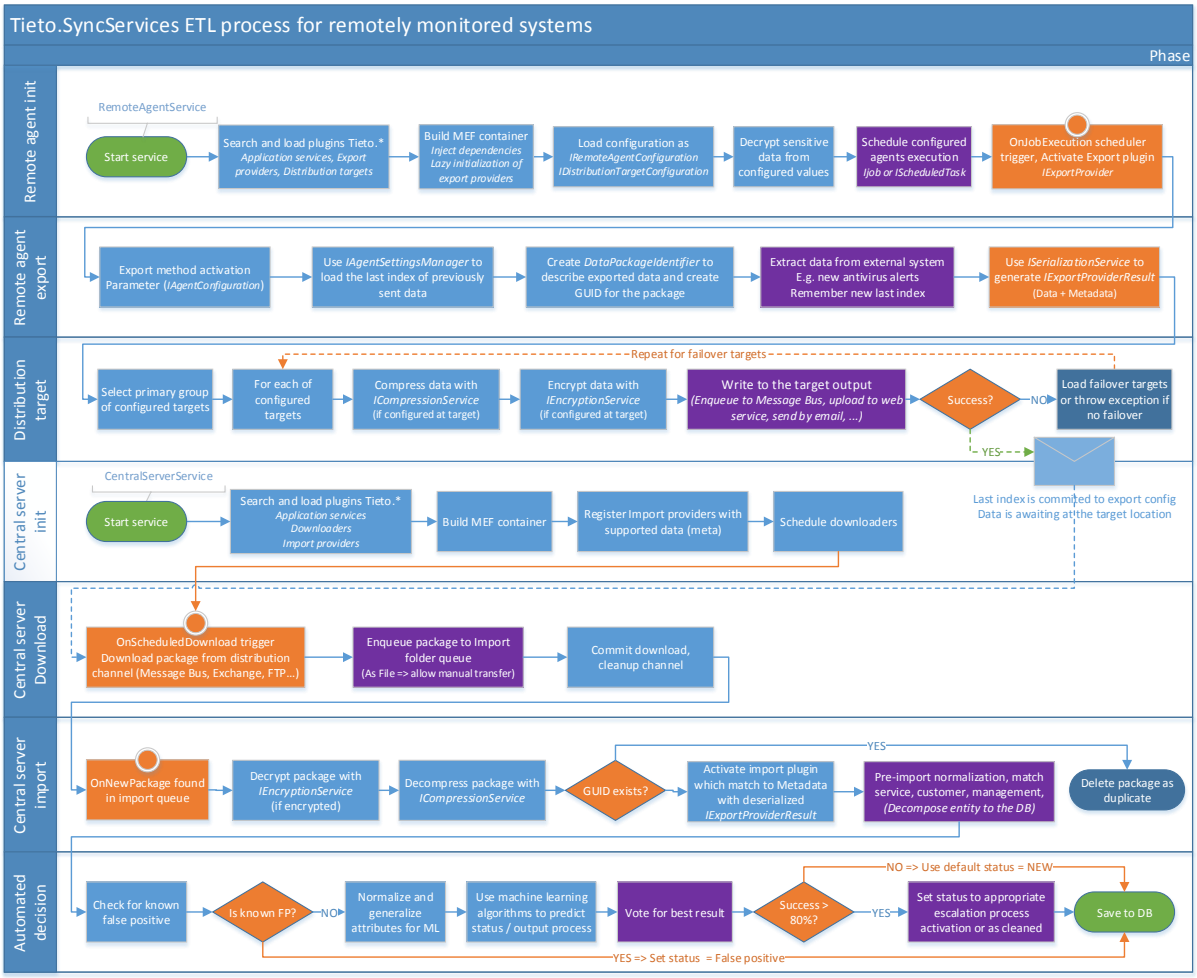
Obrázek 6: Architektura SyncServices

Základ systému tvoří dvojice Windows služeb `RemoteAgent` a `Server`, které hostují pluginy se sdílenými komponentami a exportními kódy. Funkce společné i pro jiné aplikace pak jsou implementovány v knihovně `Tieto.Common` a publikovány přes privátní NuGet[17] server. Plugin architektura je založena na MEF a jednotlivé dll knihovny jsou načteny pomocí Module katalogu.

[16] ETL (Extract-Transfer-Load) proces je možné zajistit i jinými cestami než vlastní implementací, např. jednoduchými MS SSIS balíčky nebo velkým řešením v podobě BizTalk či jiného EAI produktu.

[17] NuGet je platforma pro sdílení a instalaci knihoven pro projekty ve Visual Studio `https://www.nuget.org/`

Na obrázku 7 je zobrazen celý ETL proces a role jednotlivých služeb systému, které budou rozebrány v následujícím textu (*v rámci úspory místa jsou použity anglické popisy*).

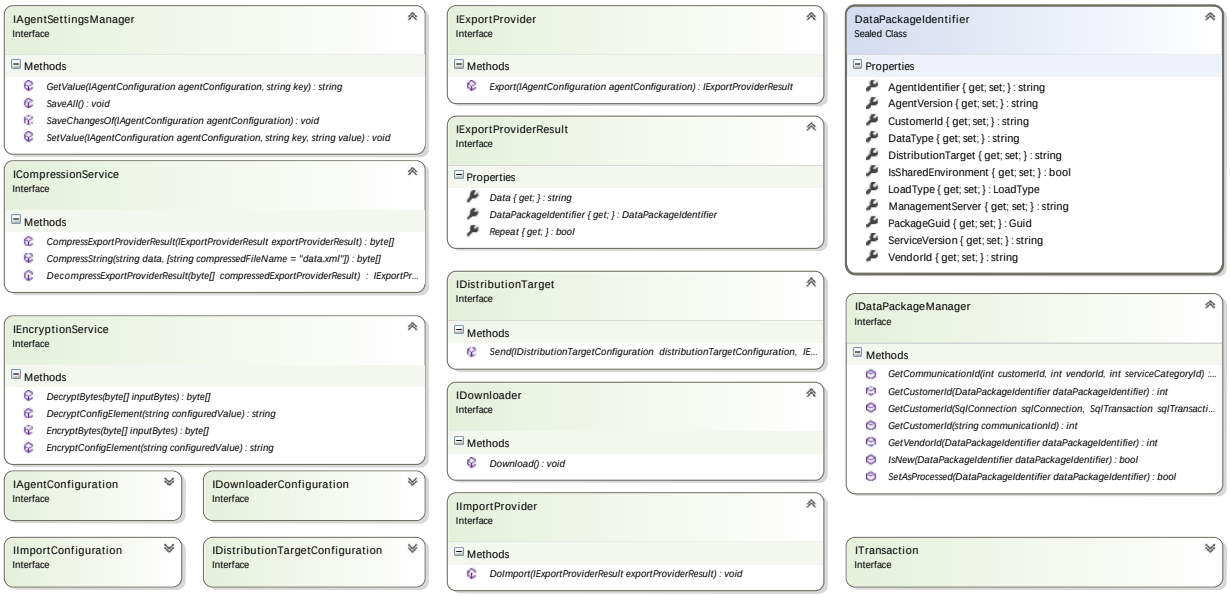


Obrázek 7: Tieto.SyncServices ETL proces

5.1.1 Jádro služeb

Je tvořeno několika singleton třídami, které jsou systému nabídnuty přes své rozhraní a reprezentují opakovanou činnost, jako je načtení konfigurace pro export/import a transparentní dešifrování citlivých údajů, načtení a uložení indexu posledního odeslaného záznamu, serializace dat, komprese a šifrování datového balíčku, plánování a spouštění jednotlivých exportů, importů nebo downloaderů, detekce duplicitních balíků, spárování se zákazníkem a výrobcem produktu aj. V diagramu 7 jsou ty nejvíce používané reprezentovány jako `IAgentSettingsManager`, `ISerializationService`, `ICompressionService` a nezakreslený `IDataPackageManager`, který řeší řadu importních procedur. Plánování je řešeno původně vlastním plánovačem, který je postupně nahrazován `Quartz.NET`. Logování je řešeno frameworkem `NLog`.

Na obrázku 8 jsou znázorněny základní rozhraní, které jsou využívány pro Export a Import dat. Další typy úloh, např. `IRemoteAction`, která provede spuštění příkazu z centrálního serveru, fungují na obdobném principu jako níže popsaný export proces, s tím rozdílem, že je aktivován na základě přijatého příkazu přes `RemoteCommandHandler`, který periodicky kontroluje frontu příkazů adresovaných na daný server.



Obrázek 8: Tieto.SyncServices hlavní rozhraní

5.1.2 Tieto.SyncServices.RemoteAgent

Windows služba, která spolupracuje se zdrojem dat, má za úkol načíst pluginy pro práci s management serverem a distribuci dat. Pluginy jsou do aplikace importovány při spuštění, pomocí MEF directory module katalogu, který prozkoumá dostupné knihovny a jejich instance přidá jako `Lazy<T>` reference do kolekce podporovaných agentů. Každá implementace se registruje spolu se svými metadaty, které určují pro jakou verzi a produkt je plugin napsán. Na základě konfigurace pak služba naplánuje spouštění jednotlivých agentů (viz výpis 4). Při aktivaci dle časového plánu pak volá příslušný exportní plugin `IExportProvider`, který má svá metadata shodná s nakonfigurovaným identifikátorem `agent=AgentIdentifier` (viz výpis 3) a pomocí metody `Export(IAgentIdentifier)` jí předá požadavek spolu s konfigurací.

Získaný výsledek `IExportProviderResult` je pak předán všem `IDistributionTarget`, které jsou pro daného agenta nastaveny v konfiguraci (viz výpis 4). Po úspěšném odeslání dat je provedeno uložení posledních indexů. Pokud provider implementuje rozhraní `ITransaction`, pak je zároveň volaná metoda `Commit` nebo `Rollback` (např. u `FileTransfer` - přejmenování souboru `_in_progress_soubor.csv` a následné smazání nebo návrat k původnímu názvu).

5.1.3 IExportProvider

Třída implementující toto rozhraní má za úkol exportovat data z management serveru nebo obecně libovolného zdroje dat (centrální databáze, internet aj.), znormalizovat na standardní datovou entitu, serializovat a spolu s metadaty vrátit jako entitu `IExportProviderResult` hlavnímu kódu služby.

Pro každý podporovaný produkt i danou verzi (viz 5.1.7) je vytvořen samostatný Class library projekt, ve kterém jsou implementovány všechny podporované exporty (alerty, klienti, monitoring management serveru aj.).

Následující ukázka (výpis 3) ilustruje kód, potřebný pro přidání podpory exportu dat z nového systému, tudíž ze strany SyncServices jde o poměrně snadnou a pochopitelnou implementaci. Obtížnost získání samotných dat je pak dána konkrétním systémem, zda nabízí API nebo je nutné číst data z databáze, registrů, Syslog, webové služby či parsovat textové, XML, Json, HTML a další soubory.

```
[Export(typeof(IExportProvider))]
[ExportMetadata(GlobalStrings.Identifier, AgentIdentifier)] // metadata k vytvorene instanci, podle ktere je vybran spravny ExportProvider
public sealed class AlertsExport : IExportProvider
{
   public const string AgentIdentifier = "Antivirus.Alerts.McAfee.Epo.5.x";
   private readonly ISerializationService _serializationService;
   private readonly IAgentSettingsManager _settingsManager;

   [ImportingConstructor] // instance je vytvorena pomoci DI
   public AlertsExport(ISerializationService serializationService, IAgentSettingsManager settingsManager)
   {
      _serializationService = serializationService;
      _settingsManager = settingsManager;
   }

   public IExportProviderResult Export(IAgentConfiguration agentConfiguration) // metoda volana dle casoveho planu
   {
      var dataPackageIdentifier = new DataPackageIdentifier // metadata k baliku dat, sluzba k nemu pripoji jeste GUID a verzi sebe sama
      {
         CustomerId = agentConfiguration.CustomerId,
         ManagementServer = agentConfiguration.ServerName,
         VendorId = Vendor.McAfee.ToString(),
         AgentVersion = "1.0.0.0",
         DataType = GlobalStrings.Antivirus.AVAlerts,
         IsSharedEnvironment = agentConfiguration.IsSharedEnvironment,
         AgentIdentifier = AgentIdentifier
      };

      var avClients = GetAVAlerts(agentConfiguration);
      var xml = _serializationService.SerializeToXmlString(avClients, dataPackageIdentifier.DataType);
      return new ExportProviderResult { Data = xml, DataPackageIdentifier = dataPackageIdentifier };
   }

   public IEnumerable<AVAlert> GetAVAlerts(IAgentConfiguration agentConfiguration)
   {
      var startIndex = (_settingsManager.GetValue(agentConfiguration, lastIndexKey) ?? "0").ToLong(); // zjisti odkud ma pokracovat
      .... // cteni dat z databaze do vysledne kolekce
      _settingsManager.SetValue(agentConfiguration, lastIndexKey, lastIndex.ToString()); // nastavi hodnotu, commit do souboru az po odeslani
      return results;
   }
}
```

Výpis 3: Základní implementace IExportProvider

5.1.4 Přenosová vrstva

Způsob doručení a formát zpráv je inspirován dvěma známými přístupy. Prvním jsou logovací frameworky, vůči kterým v aplikaci voláme funkce Log(event) a neřešíme, jestli se událost zapíše do textového souboru, databáze, EventLogu či odešle mailem. Způsob, v jakém formátu a na jaké cíle se událost zapíše, je daná konfigurací loggeru. Druhý přístup je známý z protokolu TCP/IP a emailových serverů. Datový balík je označen identifikátorem, příjemcem a adresátem zprávy a je předán na nakonfigurované rozhraní. Stejný přístup je replikován i do SyncServices.

Po získání `IExportProviderResult` jsou tato data předána na všechny `IDistributionTarget`, které jsou asociovány s daným exportem (viz výpis 4). Systém umožňuje v případě nestabilního komunikačního kanálu (např. plánované změny sítě nebo známé problémy) odeslat balík více cestami najednou. O přijetí nebo zamítnutí se pak stará `IPackageManager` na straně centrálního serveru podle jedinečného GUID balíku.

Přenosovou vrstvou může být Tieto.MessageBus, SMTP, FTP, File system nebo jakýkoliv další způsob. RemoteAgent končí svou činnost úspěšným zapsáním dat na zvolené rozhraní. Jednotlivé cíle jsou konfigurovány tak, že je odeslání rozděleno středníkem na failover bloky a čárkou na jednotlivé cíle v daném bloku, které musí všechny skončit bez chyby, aby byl daný blok splněn. Např. `distributionTargets="messageBus; primaryMail,localfile; backupMail"` se pokusí odeslat data přes MessageBus, pokud akce selže, tak se pokusí poslat data mailem a zároveň vytvořit soubor na disku, pokud jedna z těchto akcí selže, pak pošle data přes záložní email server.

Na straně centrálního serveru pak jednotlivé `IDownloader` pluginy stahují data z této vrstvy (příjem balíků z MessageBus, emailu, stažení z FTP...) a všechny řadí do souborové fronty dle přijatých časů. Souborová fronta byla vybrána z důvodu možnosti ručního zkopírování a zařazení balíku dat pro import.

5.1.5 Tieto.MessageBus

Jednoduchý Message Bus server (viz [9]) s perzistentní frontou zpráv, který umožňuje nespojovou komunikaci v pasivním režimu mezi zákaznickou a Tieto produkční síti. Server je založen na WCF a MS SQL a je umístěn v DMZ, ze které nemá povoleno inicializovat žádné spojení do sítí s koncovými klienty. Zprávy jsou obdobně jako u emailové komunikace označeny odesílatelem a příjemcem, každá zpráva je identifikována pomocí GUID. Tímto způsobem je možné posílat jak datové balíky, tak příkazy a jejich zpětné volání s výsledky provedené akce odesílateli.

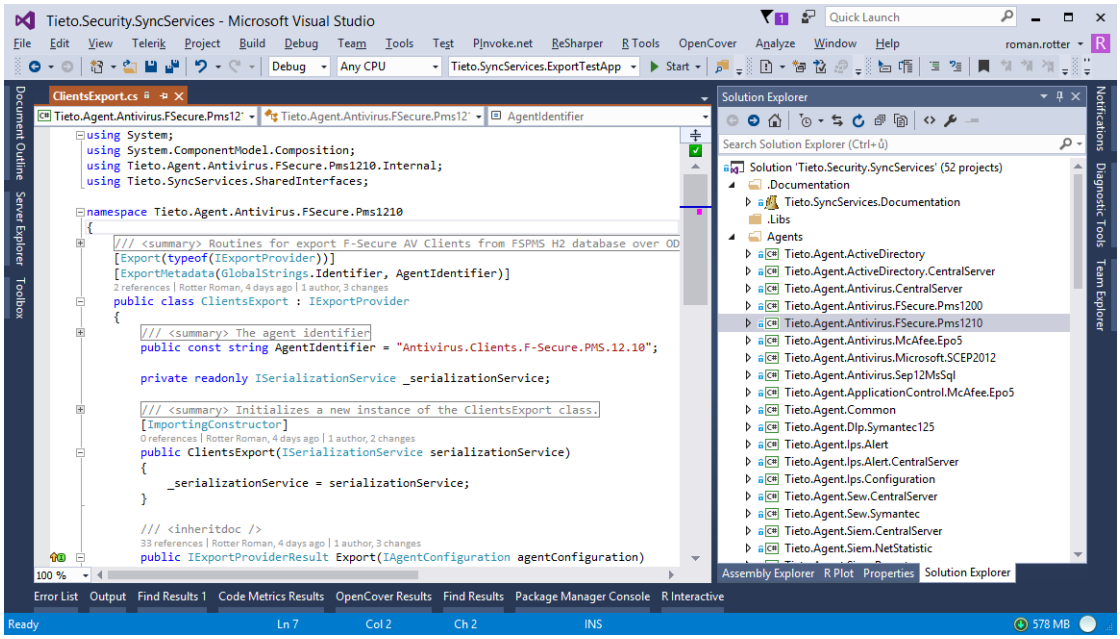
Vlastní implementace byla napsána jako výchozí komunikační server, jako koncepční řešení bylo vybráno Confluence (rozšířená Apache Kafka), které do budoucna nabízejí zajímavější možnosti škálování a směrování zpráv, ale je řešeno samostatným projektem.

5.1.6 Tieto.SyncServices.Server

Je Windows služba napojená na centrální databázi, která řeší stažení a import balíčků z RemoteAgent. Při svém spuštění načte všechny dostupné `IImportProvider` a `IDownloader`, u kterých naplánuje stahování dat z přenosových vrstev. Hlavní proces pak periodicky kontroluje adresář s frontou dat pro import. Každý zpracovávaný balík je dle potřeby dešifrován a dekomprimován. Následně si systém přečte údaje z přibalených metadat `IDataPackageIdentifier`, provede kontrolní mechanismy, zda již balík nebyl importován, jestli data pocházejí od podporovaného zákazníka atd., a poté na základě shody identifikátoru např. `DataType="AVAlerts"` aktivuje příslušný `IImportProvider`, kterému předá data a import konfiguraci.

5.1.7 Verzování export pluginů

V průběhu vývoje management serverů dochází i ke změnám v databázové struktuře nebo API, kterou je nutné implementovat, abychom mohli daný produkt nadále podporovat. V případě drobných změn by šlo problém vyřešit podmínkami, různými sadami SQL dotazů nebo jiným způsobem. Vzhledem k udržitelnosti a faktu, že starší verze produktů již nejsou vyvíjeny a očekává se jejich následné vyřazení, bylo rozhodnuto vytvářet pro každou verzi se změnou architektury nový projekt (plugin). To je ilustrováno např. na obrázku 9, kde jsou dva projekty pro F-Secure PMS 12.00 a 12.10, u kterých došlo ke změně databázového schématu, `AgentIdentifier` je konkretizován na určité verze, kde se předpokládá, že je podporována uvedená verze a všechny vyšší až po nový projekt.



Obrázek 9: Verzování export pluginů

5.1.8 Konfigurace

Konfigurace všech komponent je v systému používána přes rozhraní `IAgentConfiguration` , `IDistributionTargetConfiguration` a `IImportConfiguration`. Aktuálně jsou konfigurace implementovány pomocí XML jako vlastní `System.Configuration` sekce v rámci `app.config` souboru. Modul je aktivován na základě shodného identifikátoru, kterým je registrován v MEF. V ukázkové konfiguraci 4 je nastaven ExportProvider pro McAfee klienty z ePO 5.x, který obdrží tyto parametry jako argument své metody Export.

Po zpracování exportu předá svůj výsledek zpět export službě a ta podle ukázkové konfigurace distributionTargets odešle data do lokálního souboru a zároveň na MessageBus službu. Po úspěšném odeslání provede `IAgentSettingsManager` commit aktuálního indexu posledního odeslaného záznamu pro inkrementální data. V současnosti využívá stejný konfigurační soubor v sekci `appSettings`.

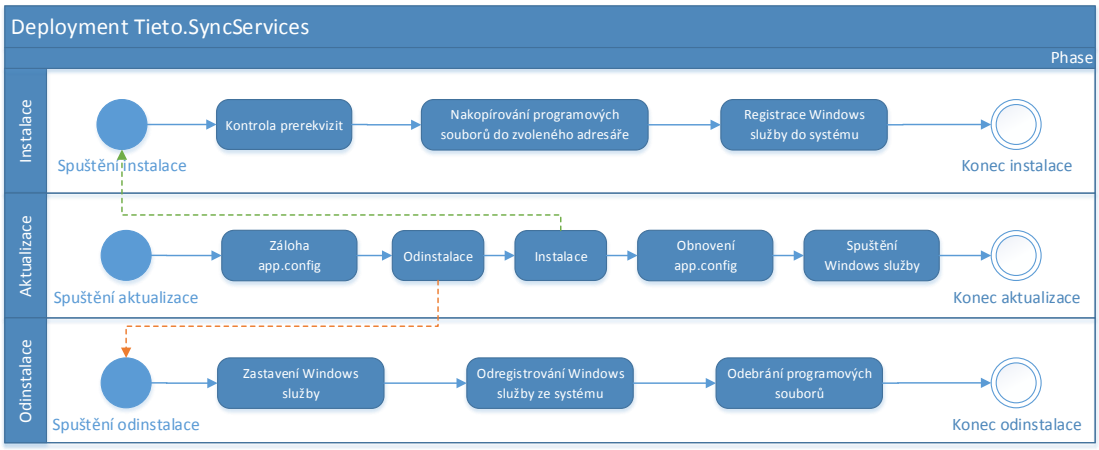
Jakákoliv citlivá část konfigurační sekce může být zašifrována pomocí AES256 přes GUI editor, kdy je šifrovaná část obalena tagem `plainText="ENCRYPTED->{AES256-Base64}"`. Při načtení config souboru jsou všechny tyto části dešifrovány v paměti a export metoda již dostává regulérní informace.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="SyncServicesAgents" type="Tieto...SyncServicesAgentSection, Tieto.SyncServices.Configuration" />
    <section name="SyncServicesDistributionTargets" type="Tieto...DistributionTargetSection, Tieto.SyncServices.Configuration" />
  </configSections>

  <appSettings>
    <add key="Antivirus.Alerts.McAfee.Epo.5.x|2|LastIndex" value="15452" />
  </appSettings>

  <SyncServicesAgents>
    <add
      key="1"
      agent="Antivirus.Clients.McAfee.Epo.5.x"
      timerInterval="00:30:00"
      customerId="Tieto"
      sharedEnvironment="true"
      serverName="TestServer"
      connectionString="Server=TestServer;Database=ePO_TEST...;Password=ENCRYPTED->{dpPbztvNA...D9rqI=};"
      databaseName="ePO_TESTSERVER"
      schemaName="dbo"
      distributionTargets="file,vqCollector" />
    <add
      key="2"
      agent="Antivirus.Alerts.McAfee.Epo.5.x"
      ... />
  </SyncServicesAgents>

  <SyncServicesDistributionTargets>
    <add name="file" provider="FileTarget" path="C:\temp\debug\" compressionOn="true" encryptionOn="false" />
    <add name="vqCollector" provider="Tieto.MessageBus" compressionOn="true" encryptionOn="true" serverAddress="10.0.1.15" port
        ="7856" />
  </SyncServicesDistributionTargets>
</configuration>
```

Výpis 4: Ukázka konfigurace export agenta

5.1.9 Deployment

Poměrně důležitým bodem je samotné nasazení těchto služeb na cílové systémy. Krom prvotní instalace je třeba počítat s průběžnou aktualizací na novější verze, která vyžaduje instalaci na 100 serverů v různých sítích, což představuje nezanedbatelný objem práce. Aplikace napsaná jako Windows služba vyžaduje registraci do systému, buď pomocí instalátoru, nástroje `installutil` nebo s využitím metod z Windows API knihovny `Advapi32.dll`, které by se aktivovaly argumentem z příkazové řádky, nestačí tedy pouhé nakopírování na cílový systém.



Obrázek 10: Deployment Tieto.SyncServices

Na obrázku 10 jsou znázorněny základní procesy nasazení. Všechny tyto procesy lze implementovat pomocí open source řešení WiX Toolset (viz [8]), který umožňuje napsat vlastní akce v C# nebo C++, výsledkem je MSI instalátor, který je možné spustit ručně nebo jej distribuovat a spustit bezobslužně na vzdáleném serveru pomocí některého ze server automation nástroje. Třetí variantou je možnost poslat aktualizační instalátor přes samotnou službu, která by provedla samo-aktualizaci, nevýhodou tohoto řešení je, že v případě selhání již nemáme kontrolu nad opravnou akcí, např. opětovné spuštění po restartu serveru.

5.2 Tieto.JobServer

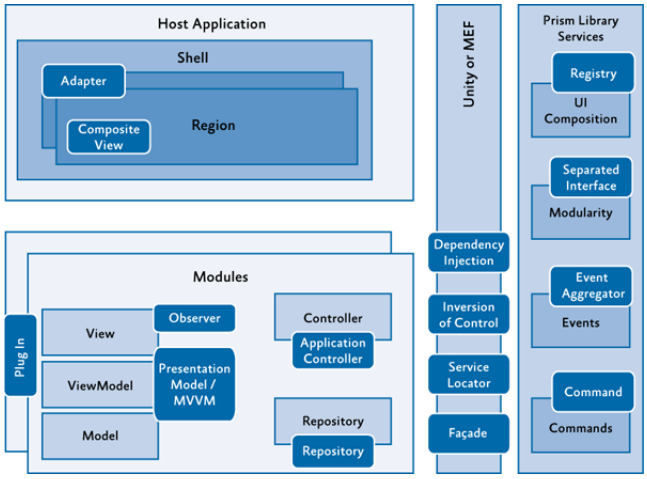
Windows služba založená na `Quartz` a `WCF` slouží pro plánování a spouštění úloh dle časových schémat, např. generování reportů, denních úkolů, automatizovaných kontrol a také jako server pro zpracování klientských požadavků, jejichž spuštění vyžaduje nebo profituje z běhu na serveru v síťové blízkosti centrální databáze (klienti jsou GUI aplikace v české síti, server je umístěn ve Finsku). V rámci této práce byla navržena pouze architektura a technologie, implementaci řešil jiný člen týmu z důvodu prioritního nasazení jiného systému a tudíž nebude tato komponenta dále rozebírána.

5.3 Tieto.Virusqueue

Je grafické rozhraní pro monitoring, konfiguraci, reportování, asset management a jiné podpůrné nástroje pro činnost Security týmu. Jedná se o modulární desktopovou WPF aplikaci, založenou na frameworku PRISM, který implementuje řadu návrhových vzorů pro WPF aplikace, DI Unity a MS SQL serveru. Formuláře jsou organizovány jako Tabbed-MDI[18] rozhraní, tj. můžeme si otevřít více formulářů ve formě záložek, včetně vícenásobného otevření stejného typu formuláře. Aplikace je dělena horizontálně i vertikálně na kompozitní bloky, které jsou vůči sobě psány přes sdílené rozhraní nebo se aktivují přes jmennou registraci do definované oblasti formuláře, tzv. Regionu. Všechny tyto bloky jsou do aplikace nahrány formou pluginů, kde z nich RegionManager, Unity kontejner a další řídící komponenty seskládají výslednou aplikaci.

5.3.1 Architektura

Základní architektura vychází z PRISM frameworku, který implementací návrhových vzorů (viz obr. 11) udává směr, jak by mohla být aplikace navržena, záleží však na vývojáři, které bloky pro svou aplikaci využije.



Obrázek 11: Návrhové vzory implementované v PRISM, zdroj [19]

V navržené aplikaci jsou tři druhy modulů. Aplikační služby, což jsou singleton třídy zpřístupněné přes sdílené rozhraní, které podporují horizontálně všechny moduly v rámci aplikace. Jedná se např. o službu pro autentizaci, navigaci, aktuální uživatel, dialogy, zoom fontů a další. Repository jsou napsány v samostatných modulech proti sdíleným rozhraním, aby byly přenositelné do jiných projektů. Samotné datové třídy (entity), jsou napsány pro přímou referenci s těsnou vazbou. GUI moduly jsou pak pouze View-ViewModel a potřebná infrastruktura.

[18] Multi-document-interface: aplikace pracující s více instancemi jednoho dokumentu (formuláře), typicky např. moderní webový prohlížeč.

5.3.1.1 Konvence

Oproti bakalářské práci byly rozšířeny konvence, které nahrazují zbytečné rozhraní, sloužící např. k aktivaci View pro standardní formuláře ze sekcí jako je Monitoring, Administrace a Reporting, např. `IMonitoringView<Entity>`. Položka v menu je pak spárována s rozhraním pomocí makra `Typ#Entita`, např. `Monitoring#Domain.Common.News`, při vyvolání Command se pak makro přeloží na řetězec, kterému rozumí reflexe a následně umí DI kontejner nalézt správnou implementaci.

5.3.1.2 Repository

Vrstva, která se stará o persistenci entit. V této aplikaci se pro práci nad velkými tabulkami od metody pro čtení z databáze vyžaduje, že informuje o průběhu načítání dat přes rozhraní `IProgressInfo`, dovede pomocí `CancellationToken` zrušit operaci čtení (např. uživatel zadá chybný filtr a následuje čtení mnoha tisíc záznamů) a je rychlá i přes síťové zpoždění dané infrastrukturou (Finsko-Česko, $delay \geq 70ms$). Namísto ORM využíváme vlastní generátor C# kódů i SQL dotazů dle databázového schématu, založeném na konvencích a extension metody redukující standardní C# kód. Náročnější monitorovací dotazy jsou psány a laděny ručně s optimalizací na straně databáze. Repository automaticky překládá UTC datum na lokální čas a zpět, takže v aplikaci pracujeme s lokálním časem, zatímco všechny časové údaje jsou ukládány v UTC. Dále nastavuje příznak, že je entita uložena a také hodnotu property, podle které poznáme, že je načtena celá entita nebo jen některé z jejich atributů, popř. pouze atribut primárního klíče. Repository jsou psány formou modulu v projektech `Tieto.Modules.Repositories.*`

5.3.1.3 Entita

Datová třída dědící z `EntityBase`, která implementuje `INotifyPropertyChanged` a `INotifyDataErrorInfo`, čímž notifikuje vrstvy nad ní, že došlo ke změně hodnoty property, a také informuje, zda je daná entita validní. Pokud ne, tak identifikuje, která property je chybně nastavená a jaký problém byl nalezen. Vyšší vrstvy reagují podle napsané logiky, v případě View překreslením hodnoty na formuláři dle aktuálního stavu, u nevalidních entit červeným rámečkem, vykřičníkem nebo jiným způsobem. Ve ViewModelu pak navázáním na OnPropertyChanged reflektujeme stav např. na SaveCommand, viz kapitola 5.3.2.1. Implementace rozšiřuje (z kopie kódu) původní PRISM `BindableBase` o sledování stavu, zda je entita uložena pomocí property IsDirty a textovým indikátorem "*"pro GUI. Hodnotové `SetProperty` rozšiřuje o sledování referenčních property z tříd složených kompozicí pomocí `SetChildTrackingProperty` a různé utility jako nastavení dle jiné instance, vygenerování SQL parametrů ze svých hodnot apod. Entity se nacházejí v projektech `Tieto.Domain.*`.

5.3.1.4 View-ViewModel

Reprezentují grafickou vrstvu (View) a její podpůrnou vrstvu (ViewModel), ve které je napsána veškerá logika a drženy stavy proměnných (kolekce a další prvky, na které je navázáno GUI). View využívá nižší vrstvu přes Binding, tj. sváže hodnotové property a commandy s grafickými prvky. Každé View (formulář) pak může být rozděleno na regiony, do kterých PRISM automaticky nahraje takové View, které je pro daný region registrováno nebo se na něj odkazuje přes Navigation Target. View-ViewModel je psáno pro každou problémovou doménu zvlášť v modulech `Tieto.Modules.UI.*`.

5.3.1.5 Aplikační služby

Vše co dovede horizontálně podpořit celou aplikaci, nezávisle na daném modulu, je napsáno formou služeb. Služba je třída, registrovaná v DI kontejneru jako Singleton. Může, ale nemusí, záměrně využít výhody singleton instance, např. globální změna zvětšení písma. Služby musejí být navrženy pro přístup z více vláken. Moduly pro aplikační služby jsou definovány v `Tieto.Modules.Services.*`.

5.3.1.6 Sdílení knihoven

U znovupoužitelných komponent bylo nutné vyřešit i sdílení jejich různých verzí v rámci více Solution (projektů). Bylo otestováno několik řešení, obvykle v rámci Team Foundation Server, nakonec byl na náš interní server nasazen privátní NuGet. Přes něj si mohou programátoři stáhnout jak obecné knihovny Tieto.Common a Tieto.Wpf, tak repository pro problémové domény, sdílené rozhraní, datové entity aj., přitom je zde nativně vyřešeno i verzování a aktualizace použitých knihoven. Můžeme tak např. využít stejné repository jak pro GUI, tak pro JobServer. Celý NuGet server je ke stažení jako balíček `NuGet.Server`, který se připojí ke standardnímu web projektu, nasazení tak bylo otázkou necelé hodiny. Do budoucna je možnou alternativou Package management, který byl přidán do TFS2017, aktuálně je však využívána starší verze.

5.3.1.7 Styly a ikony

Jeden z nespecifikovaných požadavků, který se ale předpokládá u všech aplikací, je konzistentní vzhled formulářů. Výhodou WPF je možnost definovat styly včetně ikon, podobně jako je to běžné u webových aplikací. Základní styly jsou definovány v projektu `Tieto.Themes.Virusqueue`, kde jsou jak barvy, tak základní ikony, které aplikace využívá přes klíče. Globálně tak můžeme měnit jak samotné obrázky, tak barevné schéma. Barvy entit, jako je nebezpečnost malware, jsou zatím pracovně v databázi definovány konkrétním kódem (red, blue, green...), převod na abstraktní barevnou paletu bude řešen později, kdy bude jasnější, jak velkou škálu je třeba použít. Do té doby budou uživatelé ochuzeni o možnost přepnutí tématu např. ze světlého na tmavé, které méně zatěžuje oči na nočních směnách.

5.3.2 Správa uživatelů a zabezpečení

Přístup uživatelů je řešen dvoufázově, pomocí Active Directory integrovaného ověření vůči SQL serveru na základě role a následně ověřením uživatele a jeho oprávnění v tabulce `Common.User` Uživatelé jsou do systému importováni z Active Directory, ruční založení záznamu aplikace záměrně nepodporuje. Automaticky je importován SID uživatele, takže si aplikace může kdykoliv ověřit aktuální informace v AD. Uživatel je přiřazen do jedné nebo více rolí, přes které jsou nepřímo definovány jeho práva v rámci aplikace. Jediná úroveň oprávnění, která je vázána přímo na uživatele, je vlastník záznamu, což se však v části pro antiviry nevyužívá.

5.3.2.1 Oprávnění

Každý formulář má napevno přiřazen globální identifikátor GUID, pomocí kterého se pak v administraci nastavuje úroveň oprávnění. Pokud uživatel nemá na formulář přístup ani pro čtení, pak se tato položka nezobrazí ani v menu. Stejná technika je využita pro libovolný objekt nebo akci, pro který chceme řídit oprávnění. Všechny objekty jsou uloženy v tabulce Common.SecurableObject a definují GUID, název a případně AssemblyFullName. Aplikační služba `IAuthorizationService` pak pro požadované GUID vrátí úroveň oprávnění, podle kterého napíšeme příslušný obslužný kód.

Ve výpisu 5 je konkrétní ukázka, jak v editačním formuláři povolit nebo zakázat všechny GUI prvky vztažené k akci `Delete` (tlačítko, menu, klávesovou zkratku....). Metoda `ICommand.CanExecute()` testuje, zda je možné daný command spustit. Obvykle zde jsou různé funkční podmínky, u metody save např. validace entity, u asynchronních commandů zda nejsou aktuálně spuštěny. Zde na úrovni ViewModelu v metodě `CanDeleteDistributionRule()` testujeme, zda je vybrán záznam, který by bylo možné smazat a zda má uživatel oprávnění nejméně pro editaci - `_permissionLevel.CanEdit()`.

GUI prvky pak automaticky mění svůj vizuální stav na zakázané v případě, že jsme z kolekce nevybrali žádný záznam nebo když má uživatel oprávnění jen ke čtení. Metoda se volá automaticky z konstruktoru, pro její aktualizaci pak musíme zavolat `DeleteCommand.RaiseCanExecuteChanged()`, tj. zde provážeme s událostí `DistributionRulesOnSelectedItemChanged` (změnili jsme vybraný záznam).

Pokud chceme řídit např. tlačítko Save spolu s validací entity tak, že svůj stav změní přímo při úpravě (např. vypisujeme název, který nesmí být prázdný řetězec), pak provážeme `EditCommand.RaiseCanExecuteChanged()` s událostí `DistributionRulesOnSelectedItemChanged` (změnila se jakákoliv property entity DistributionRule). Test oprávnění pak automaticky zařazujeme ke všem těmto pravidlům.

```
public sealed class DistributionRuleAdministrationViewModel : DockableViewModelBase
{
   private const string Uid = "{E31074B4-5DD7-44EE-90D8-0ACAD8C04912}";
   private readonly PermissionLevel _permissionLevel;

   public DistributionRuleAdministrationViewModel(IAuthorizationService authorizationService,...)
   {
       _permissionLevel = authorizationService.GetPermission(Uid);
       if (!_permissionLevel.CanRead()) throw new UnauthorizedAccessException("You have no permission to access this form.");

      DeleteCommand = new DelegateCommand(DeleteDistributionRule, CanDeleteDistributionRule); // CanDeleteDistributionRule => metoda
   }

   // metoda ridi chovani vsech GUI prvku navazanych na DeleteCommand, napr. bez prava editace bude zakazane tlacitko delete.
   private bool CanDeleteDistributionRule()
   {
      return DistributionRules.HasSelection && _permissionLevel.CanEdit();
   }...
}
```

Výpis 5: Testování úrovně oprávnění

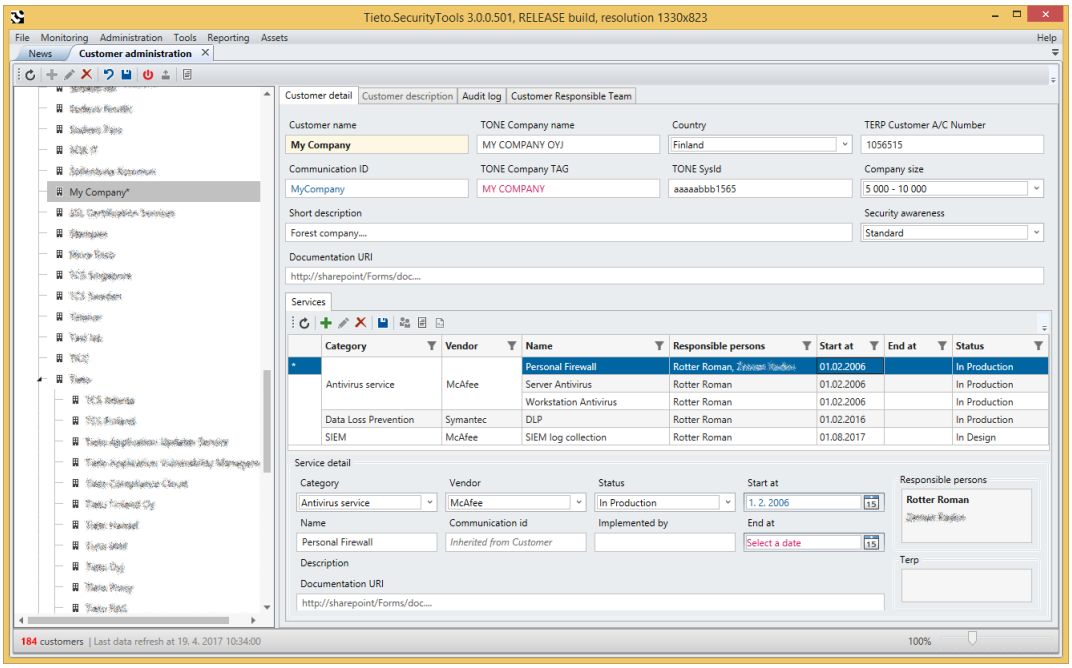
5.3.3 Správa zákazníků a služeb

Systém je navržen pro hierarchickou strukturu zákazníků, kdy si jednotlivé sub-divize firmy mohou uzavírat různé antivirové služby, např. jedna divize může mít outsourcing na stanice i servery, jiná pouze servery a mohou se lišit i jednotlivé produkty. Jsou podporovány i scénáře, kdy se firma rozdělí na více menších samostatných jednotek. Zákazník je v rámci formulářů pro produkci aktivní pouze v případě, že má pro danou kategorii platnou nejméně jednu službu. V GUI antiviru tedy uvidíme pouze ty firmy, které mají jednu nebo více antivirových služeb a jejich stav je jiný než `Retired` (zrušená). Za každou službu je zpravidla zodpovědná jedna primární osoba a jedna až více záložních. Výstupy pro uživatele jsou zpravidla formou BI reportů.

Vzhledem k rozvoji cloudových služeb a častému přechodu z dedikovaných na sdílené prostředí, bylo nutné vyřešit mapování zákazníků ze systémů, které nemají podporu pro multitenancy nebo jsou jako single-tenant záměrně nastaveny. Zákazníci jsou pak organizováni ve složkách a jejich počítače jsou do těchto složek automaticky řazeny. Pro mapování je pak využito `CommunicationId`, které odpovídá názvu složky na daném management serveru (u dedikovaného prostředí je CommunicationId definováno v konfiguraci RemoteAgent služby).

Pokud má zákazník počítače na různých prostředích, kde je nastavena různá syntaxe, pak systém umožňuje nastavit dané ID na úrovni služby, viz obrázek 12 (standardně služba dědí nastavení ze zákazníka). Samostatné sdílené prostředí je pak zavedeno jako virtuální zákazník a jeho communicationId pro vybranou službu (např. antivir) definuje šablonu, která se přeloží na regulární výraz, ze kterého zjistíme CommunicationId zákazníka např. `\\(_Production|Datacenter_test)\\Customers\\{CustomerId}\\`. Pokud zákazník s daným CommunicationId neexistuje, pak je automaticky vytvořen ve složce `~\Imported` a následně přidána informace o novém zákazní-

kovi v GUI News, viz obr. 13 v následující kapitole (pozn. vytváří se jedna zpráva pro všechny naimportované zákazníky z jednoho balíčku dat).

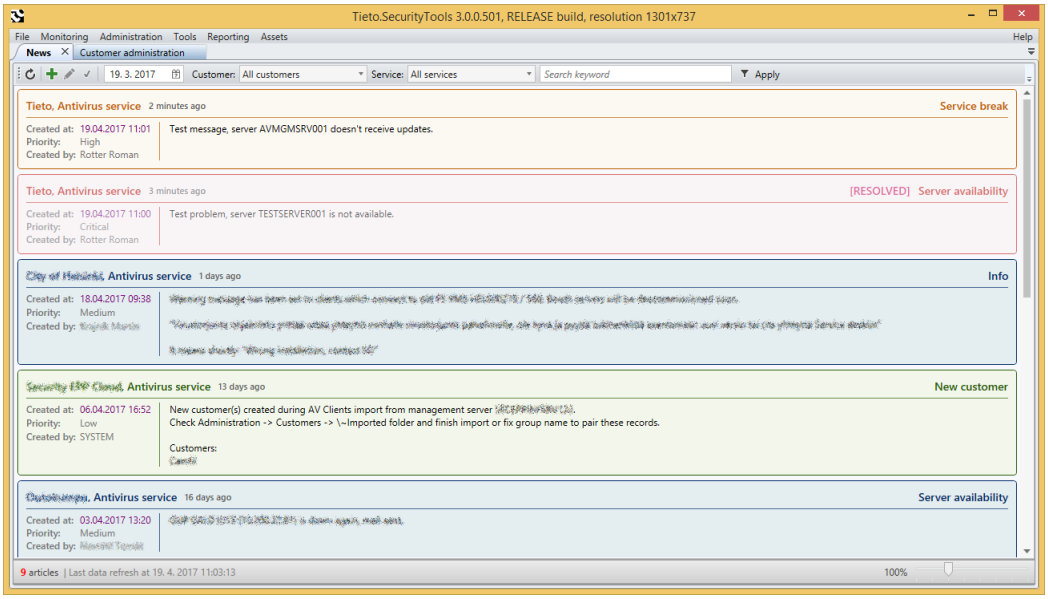


Obrázek 12: Správa zákazníků, služeb a zodpovědných osob

5.3.4 News

Při založení nového zákazníka službou Tieto.SyncServices a dalších případných zpráv vzniklých činností na straně serveru, bylo potřeba vyřešit jakým způsobem tyto informace předat (email, RSS, log atd.). Zároveň si mezi sebou předávají informace jednotlivé směny o provozních problémech (např. nedostupnosti serveru, kdy již byl daný problém eskalován), nové službě, ukončení služby pro zákazníka, výskyt anomálií apod.

Pro všechny tyto případy byl navržen modul News, který přehledně koncentruje všechny tyto informace, zároveň je možné vyhledávat ve zprávách podle klíčového slova. Zprávy jsou kategorizovány a barevně odlišeny podle závažnosti, vyřešené problémy jsou označeny jako [RESOLVED] se simulací potlačení pomocí průhlednosti, viz 13, zprávy ze serveru jsou identifikovány uživatelem SYSTEM.



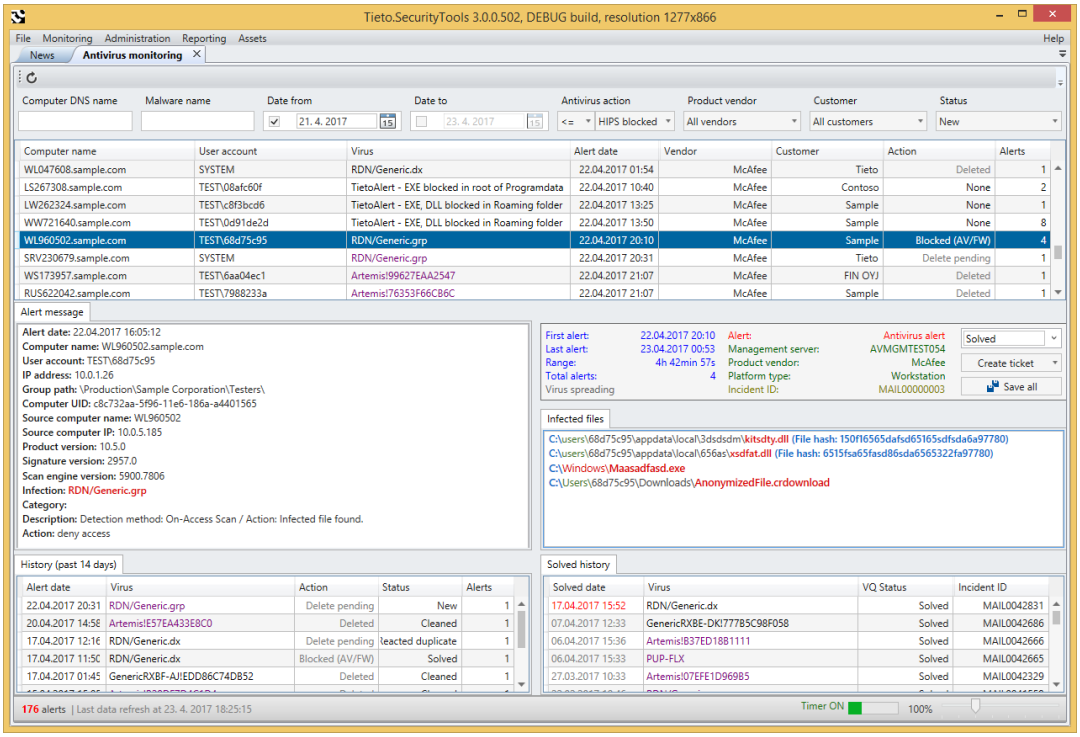
Obrázek 13: Rozhraní pro sdílení provozních informací.

5.3.5 Antivirus monitoring

Grafické rozhraní pro monitorování antivirů je navrženo minimalisticky, ale zároveň tak, aby měl operátor hlavní informace na jedné ploše, viz obrázek 14. Při běžné práci jsou vyfiltrovány nevyřešené alerty (vybraný záznam na obr. byl eskalován, má referenční číslo a po uložení bude odfiltrován). Jejich seznam je zobrazen formou tabulky, navázanou na kolekci hlavičkových atributů alertu, která obsahuje základní údaje (méně atributů = rychlejší přenos dat z databáze). Zbylé údaje z kompletního alertu, se pak dynamicky načítají až při vybrání určitého záznamu, a to z důvodu, aby byla aplikace dostatečně rychlá i při výběru většího počtu alertů.

Pod seznamem alertů je GUI rozděleno na textový detail alertu a čtyři regiony, které přebírají informaci o vybraném alertu pomocí RegionContext a představují základní pohledy nutné k rozhodování. Každý z nich je řešen vlastním View-ViewModel, registrovaným pro daný region a využívá jinou metodu v AntivirusRepository. Sdílením dat v regionech a reakcí na jejich změnu, spolu s asynchronními metodami, se všechny detailní data načítají nezávisle na sobě a jsou spuštěny paralelně v okamžiku změny záznamu pouze na základě změny RegionContextu. Do budoucna se tak dá tento formulář rozšířit o další informace, např. grafický průběh za určité období, dodatečné statistiky aj., bez zásahu do hlavního ViewModelu.

Formulář využívá schopnosti oka vyhodnocovat barevnou informaci. Základní zvýraznění je použito v tabulkách, kde např. TrackingCookie je šedou barvou, testovací alert zelenou, standardní černou, heuristika fialovou a nebezpečný malware pak červeně, případně vysoce kritický může být zvýrazněn i změnou fontu. Stejně tak jsou rozlišeny akce antiviru nebo datum, kdy byla v minulosti na vybraném počítači, řešena infekce. Červeně označujeme situaci, kdy byl počítač



Obrázek 14: Rozhraní pro řešení alertů (citlivé údaje anonymizovány).

řešen před třemi a více dny a očekává se, že by neměl mít žádnou infekci. Zeleně pak období, kdy byl předchozí alert eskalován, ale je ponechán čas na vyčištění a šedě potlačíme historicky velmi staré údaje. V historii řešených událostí k danému počítači zobrazujeme údaje za poslední rok, aby bylo zřejmé, že se jedná např. o často infikovaný počítač, který by bylo vhodné přeinstalovat a zaškolit uživatele.

Detailní výpis z alertu a seznam infikovaných souborů je transformován na RTF dokument, ve kterém jsou programově zvýrazněné důležité údaje. Především se jedná o zvýraznění části cesty v infikovaném souboru, kde operátor snadno vidí, na kterém disku se infekce nachází, známé adresáře první úrovně mají své barevné kódy, např. systémový je červeně, zatímco Users zeleně. Zvýraznění samotného názvu souboru tučnou červenou barvou snadno zviditelníme, o který infikovaný soubor se jedná, stejně jako jeho hash.

Tlačítková lišta a oblast pod filtrem jsou ponechány pro další rozšíření o advanced filtr (v expanderu), ad-hoc analytické pohledy, vyhledání informací na VirusTotal a jiné funkce, které nejsou předmětem této práce.

Pokud se operátor rozhodne, že nebude daný alert eskalovat, pak změní jeho status na odpovídající hodnotu a uloží záznam. V případě eskalace využije tlačítko `Create ticket`, což je `SpinButton`, který volá jedinou metodu s parametrem, rozlišujícím typ eskalace. Pro Sample request nebo Escalate slouží šipka na kraji tlačítka, kterou vybere jinou než výchozí akci.

Celý systém je navržen tak, aby se namísto koncentrace složitého kódu, zřetězilo využití několika jednodušších, univerzálních modulů a proto i v tomto místě končí (co se týče programového kódu), veškerá další programová logika. Pomocí implementací, schovaných za rozhraní `ITemplateSelector` a `IDistributionService` zajistíme, že systém navrhne instrukce a zobrazí GUI pro správnou eskalaci spolu s kontakty. Operátor má možnost před vytvořením ještě upravit instrukce nebo kontakt. Pokud se podíváme na výpis 6, tak jediným úkolem v metodě CreateTicket je předat daným službám model (vybraný alert), druh eskalace a callback akci, která se provede po eskalaci. Důležité pro tento ViewModel je, že jsme problém "předali" k vyřešení a daná služba nám již pouze vrátí referenční číslo emailu, ITSM tiketu aj., které si pro daný alert uložíme.

Dále si můžeme všimnout, že zde má operátor řadu alertů, jejichž detekované soubory antivir zablokoval nebo smazal.

V následujících kapitolách se zaměříme na to, jakým způsobem pomocí strojového učení automaticky odstranit alerty, které nevyžadují eskalaci z monitoringu a také jak správně vygenerovat instrukce pro libovolný problém, nalézt správné kontakty a následně jej pomocí integrací eskalovat na zodpovědné osoby.

```
public sealed class AntivirusMonitoringViewModel : DockableViewModelBase
{
   private readonly ITextTemplateSelector _textTemplateSelector;
   private readonly IDistributionService _distributionService;
   ...

   public AntivirusMonitoringViewModel(ITextTemplateSelector textTemplateSelector, IDistributionService distributionService, ...)
   {
      _textTemplateSelector = textTemplateSelector;
      _distributionService = distributionService;
      ...

      CreateTicketCommand = new DelegateCommand<object>(CreateTicket, CanCreateTicket);
   }

   public DelegateCommand<object> CreateTicketCommand { get; }
   ...

   private void CreateTicket(object parameter)
   {  // zjisti kategorii z parametru SpinButton Create ticket - StandardRequest, Escalate, SampleRequest
      var categoryCode = (DistributionRuleCategoryCode)parameter;
      // podle kategorie vyhleda kod statusu, se kterym se alert ulozi (preklad enum => enum)
      var resultCode = GetAlertStatusCode(categoryCode);
      // Sluzba TemplateSelector navrhne vhodne instrukce na zaklade modelu AlertDetail a kategorii
      var instructions = _textTemplateSelector.SelectAndRender(AlertDetail, (int)ClassIdentifierCode.AntivirusAlert, categoryCode, ...);
      // Eskalace napr. odeslanim emailu, vytvorenim ITSM ticketu. Pomoci callback akce nastavi vracene referencni cislo, napr. MAIL015605056
      _distributionService.Prepare(AlertDetail, categoryCode, new Instruction(instructions), referenceNumber =>
      {
         Alerts.SelectedItem.ReferenceNumber = referenceNumber;
         Alerts.SelectedItem.Status = Statuses.FirstOrDefault(status => status.StatusId == resultCode);
         if(resultCode == AlertStatusCode.UnderInvestigation) _antivirusAlertRepository.AddToFalsePositives(AlertDetail);
      });
   }

   private bool CanCreateTicket(object parameter)
   {
      return _permissionLevel.CanEdit() && Alerts.HasSelection && Alerts.SelectedItem.AlertId == AlertDetail?.AlertId;
   }
}
```

Výpis 6: Eskalace z AntivirusMonitoringViewModel

6 Popis automatizovaných úloh a metod strojového učení použitých k jejich realizaci

V této kapitole se zaměříme na části, které je potřeba na základě popsaného procesu automatizovat. Následně si popíšeme algoritmy, kterými je možné tyto problémy řešit a způsob, jakým je tato automatizace ve výsledku implementována. Výběr konkrétního klasifikačního algoritmu pak bude proveden na základě testů z kapitoly 7.

6.1 Automatizovaná klasifikace - filtrace alertů

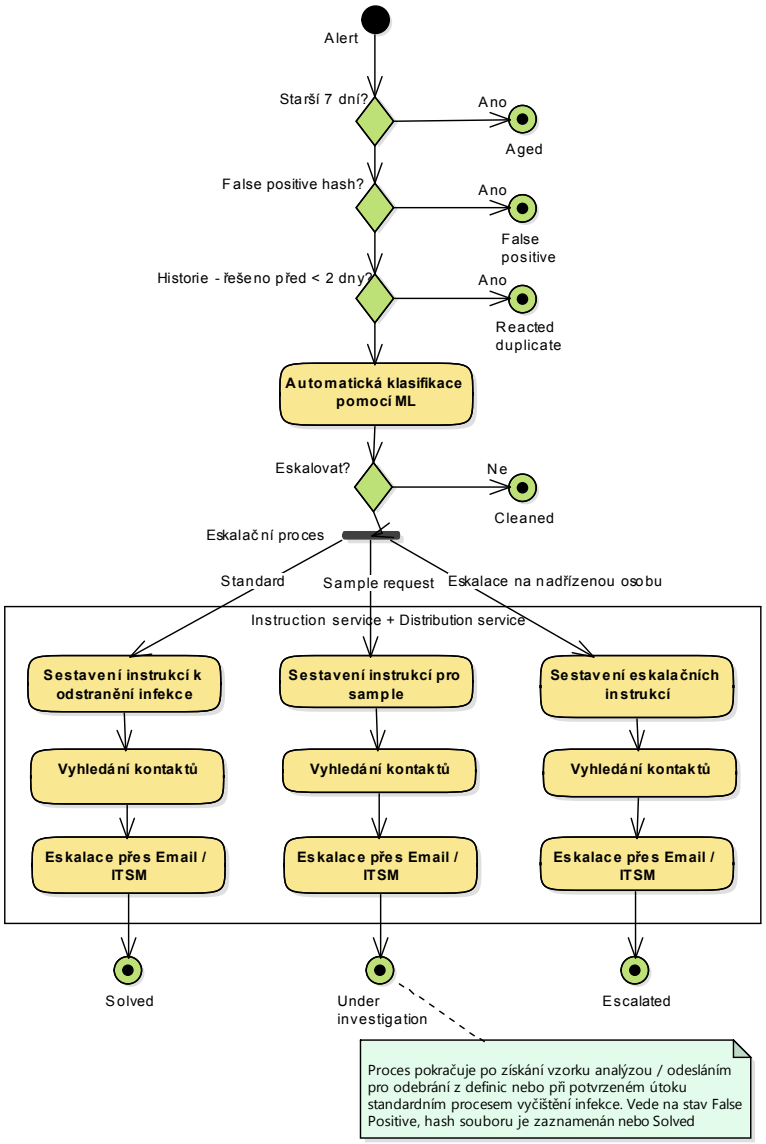
Proces zpracování alertu definuje řadu kontrol a pravidel, jak má být daný problém eskalován. U některých zákazníků i speciálními podprocesy. V této práci se zaměříme pouze na výstupy, které jsme schopni podpořit automatizací. Očekávané výstupy z procesů jsou:

- **Cleaned** - akci antiviru považujeme za úspěšnou, nevyžadujeme další eskalaci.
- **Accepted Risk** - riziko je zanedbatelné, např. Tracking Cookie nebo se jedná o schválené rizikové aplikace (nepodepsané interní nástroje nebo admin utility - nmap, pskill apod.)
- **Aged** - alert byl vygenerován v době, kdy byl zákazník mimo síť, např. na služební cestě a nemá smysl zpětně tuto infekci řešit (při přetrvávajícím problému je zpravidla po připojení k síti vygenerován nový alert).
- **False positive** - byla rozpoznána chybná detekce čistého souboru, např. antivir smaže klienta podnikového informačního systému. V tomto případě budeme žádat o vzorek malware, abychom zajistili jeho vyjmutí z detekcí a dočasným nastavením výjimek, než bude tato detekce opravena výrobcem antiviru.
- **Solved** - ze strany security týmu byly sepsány instrukce k odstranění malware a předány zodpovědnému týmu - tzv. Eskalace.
- **Reacted duplicate** - v případě opakovaného alertu, který již byl eskalován před definovanou časovou hranicí, se další eskalace nevytváří.
- **Escalated** - infikovaná stanice byla řešena více než dvakrát, ale počítač je opakovaně infikován. V rámci tohoto procesu jsou buď poslány striktnější instrukce, nebo je problém eskalován přes nadřazené osoby (regionální security manažery apod.).

Obecně budeme vycházet z faktu, že binární klasifikace je výpočetně méně náročná a umožňuje využít širší spektrum algoritmů s lepšími výsledky než multiclass klasifikátory, a proto se budeme snažit omezit klasifikaci na binární (viz [10, kapitola 7.6] a dokumentace ke knihovnám).

První fází filtrace je transformace alertů na incidenty, kdy seskupíme n alertů, generovaných z jednoho počítače za 24 hodin na $N : M$ relaci, kde N budou tzv. incidenty obsahující agregovanou informaci o počítači a produktu + malware + akce. M pak budou relační entity představující infikovaný soubor, hash a počet výskytů. Počet relací M ztrátově omezíme na $M \leq threshold$. Od dané hranice již nové soubory zahazujeme a pouze zvyšujeme čítač (další již nepřinášejí novou informaci). Statistický průměr za rok odpovídá redukci 87% příchozích alertů.

Dále následuje automatizované rozhodování, které alerty není nutné řešit, což odpovídá další redukci o 76%. Bezpečnostní analytik se tímto může soustředit pouze na reálné problémy, které odpovídají pouhým 3% celkového objemu vygenerovaných alertů, což činí necelých 60 000 incidentů za rok 2016.



Obrázek 15: Podpora řešení alertu (fáze po agregaci)

Následně podpoříme jednotlivé procesy tak, aby bylo možné minimalizovat nebo eliminovat manuální činnost. Úlohou analytika je pak rozhodnout, jak daný problém řešit a v případě potřeby upravit instrukce. Aktuálně se indicenty, které vyžadují řešení, zpracovávají podle třech podprocesů, jež budou podporovat také moduly připravující instrukce k řešení problému a vytvoření eskalačního požadavku.

Dle diagramu na obr. 15 si problém po prvotní agregaci dekomponujeme na několik částí. V případě shody ukončíme proces s daným statusem:

1. V první fázi odstraníme Aged alerty, které lze filtrovat na bázi data.
2. Na zbylých alertech provedeme test na známý hash souboru v tabulce false positive.
3. Problém, který byl eskalován předchozím alertem v čase t - *threshold*, uzavíráme jako Reacted duplicate (např. 2-3 dny u stanic).
4. Následně zobecníme status "Cleaned" vs "Vše ostatní", čímž dostaneme binární klasifikační problém. Ten budeme řešit pomocí algoritmů z kapitoly 6.2.
5. Zbylé problémy jsou rozhodnutelné analytikem, aplikace podpoří tyto procesy automatizací instrukcí a eskalačním engine:
    - Solved - sestaví se instrukce k řešení dané infekce, problém je uzavřen eskalací přes email nebo ITSM.
    - False positive - vyžádáme si vzorek daného souboru s instrukcemi, jak postupovat k jeho získání a bezpečnému předání pomocí interního nástroje. Po obdržení následuje analýza a v případě chybné detekce eskalace na výrobce k odstranění z definic. Systém si v tomto případě uloží hash a označuje stejné soubory jako False Positive.
    - Escalated - sestaví instrukce k řešení, obsahující informaci o neúspěšných pokusech a odešle na nadřízené osoby / tým.

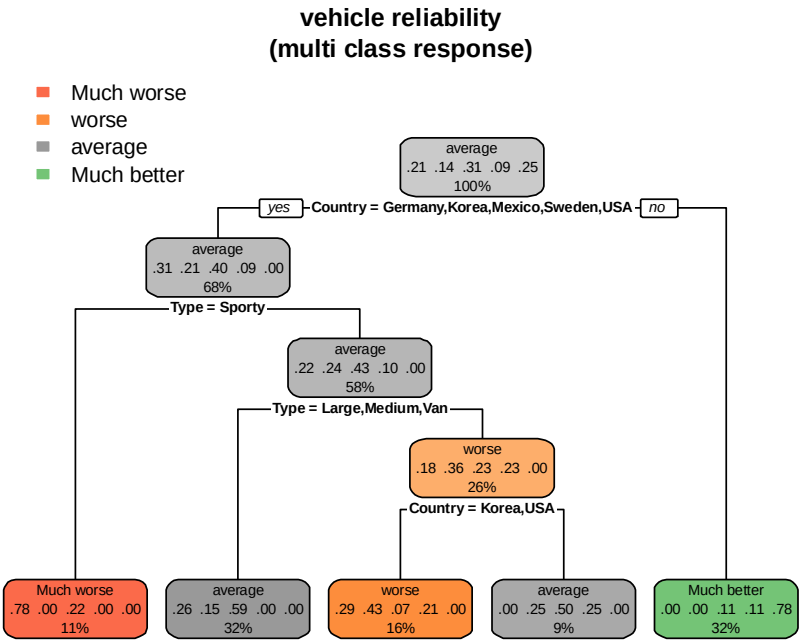
V další fázi produkčního nasazení pak pomocí atributu "Solved with default instructions", který nastaví analytik při eskalaci, rozlišíme problémy, které nevyžadovaly žádnou úpravu vygenerovaných instrukcí a kontaktů. Stejným způsobem, jakým v této práci vyřešíme klasifikaci alertů, které nepotřebovaly eskalaci, pak můžeme rozlišit případy, které odešleme s vygenerovanými instrukcemi automaticky.

6.2 Klasifikační algoritmy

V této práci řešíme klasifikační problém, kdy máme k dispozici trénovací množinu s kategoriálními daty a automat má rozpoznat, zda daný problém vyžaduje řešení nebo se může infekce označit jako vyčištěná. Dále je třeba implementovat detekci anomálií, kdy musí systém rozpoznat, že úroveň alertů, která kolísá na dni v týdnu a konkrétní hodině, odpovídá běžnému chování nebo se jedná o anomálii (false positive nebo virový outbreak). Tvorba modelu a testování algoritmů je rozebrána v samostatné kapitole 7.

6.2.1 Decision Tree

Česky "Rozhodovací strom" je učící metoda vhodná jak pro klasifikaci do dvou a více tříd (viz obr. 16), tak regresi. Strom reprezentuje hierarchickou skupinu podmínek, které dělí množinu do konečného počtu tříd. Každý uzel stromu dělí množinu na dva a více podproblémů, listy reprezentují výslednou třídu (popř. hodnotu u regresního stromu). Algoritmy založené na rozhodovacích stromech se umí vypořádat jak s šumem, tak chybějícími hodnotami. Aktuálně existuje přes deset různých algoritmů, nejpoužívanější jsou C4.5, C5 a CART, princip konstrukce stromu bývá obvykle demonstrován na ID3 algoritmu.



Obrázek 16: Ukázka rozhodovacího stromu, zdroj `http://www.milbo.org/rpart-plot/`

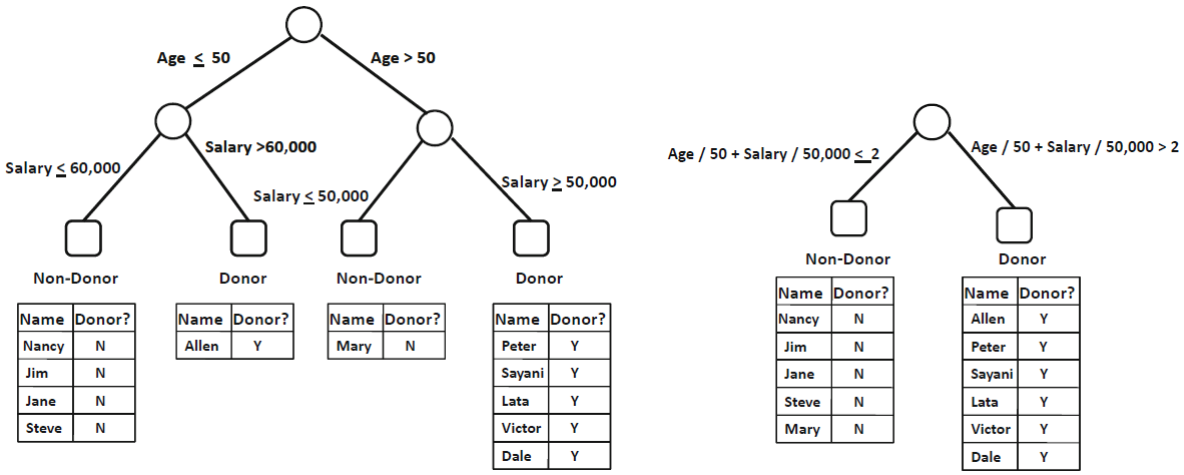
Detailní popis, jak tento algoritmus funguje, nalezneme v knize věnované rozhodovacím stromům [15], rozsáhlý popis lze také nalézt v knihách [13, kapitola 4] a [14, kapitola 8]. Zde se seznámíme s klíčovými vlastnostmi, následující text je parafrázován z literatury [12].

6.2.1.1 Sestavení stromu

Strom je sestaven hierarchickým *top-down* clusteringem, kdy v každém uzlu hledáme tzv. *best-split*, neboli nejvhodnější kritérium pro další dělení v závislosti na požadovaném výstupu z podmnožiny dat trénovací množiny, získané dělením z nadřazeného uzlu.

6.2.1.2 Split - větvení stromu

Cílem výpočtu v každém uzlu je co nejlépe rozdělit odlišné třídy v podřízených uzlech, tzv. *best-split*. Podle počtu atributů rozlišujeme split na *univariate* (jeden atribut) nebo *multivariate* (více než jeden atribut), viz obr 17. Multivariate split nevytváří zbytečně hluboké stromy, do dalších uzlů postupuje více vzorků, ze kterých počítáme další split a z tohoto důvodu je schopen lépe potlačit chyby či šum v trénovací množině.



(a) Univariate split

(b) Multivariate split

Obrázek 17: Univariate vs Multivariate Split, zdroj [12]

Výpočet samotného *split kritéria* závisí na druhu atributu, který může být:

- **Binární** - existuje jen jediný možný split - ano/ne, kde dochází k rozdělení na dvě větve.
- **Kategoriální** - pokud má atribut r různých hodnot, pak může být buď vytvořeno r větví, nebo se můžeme pokusit seskupit atributy tak, abychom dosáhli binárního větvení, což vyžaduje otestování $2^r - 1$ kombinací kategoriálních atributů a sestavení adekvátního pravidla (jeden nebo skupina atributů = true vůči zbytku). Alternativní možností je převod kategoriálních hodnot na binární data, viz [kap. 2] [12]
- **Číselný** - pokud atribut obsahuje malý počet r diskrétních hodnot, pak je možné vytvořit r podvětví pro každou diskrétní hodnotu zvlášť. U spojitých hodnot nebo větším počtu diskrétních hodnot se využívá split kritérium $x \leq a$, kde x je hodnota atributu a a je vypočtená konstanta.

Pro výpočet nejvhodnějšího kritéria rozdělení množiny v uzlu N, tzv. *best-split*, existuje několik metod. Pro matematické vyjádření si označíme S jako trénovací množinu, která vstupuje do uzlu, a $S_1, S_2 ... S_r$ jako podmnožiny z trénovací množiny, vzniklé větvením uzlu na r pod-uzlů. p pak bude pravděpodobnost výskytu hodnoty atributu p_j vůči cílové třídě a jako k označíme počet tříd daného atributu (např. Obloha $\in$ {slunečno, zataženo, déšť } ... k=3).

- **Error rate** (Misclassification) - vypočítává pravděpodobnost, že ohodnocení vůči trénovací (pod)množině bude chybné. Jako kritérium je zvolena vlastnost s nejmenší chybou.

$$Error(S) = 1 - \max_j p_j$$

- **Entropie / informační zisk** - spočteme hodnoty entropie pro každý atribut v podmnožině trénovací množiny. Následně spočteme informační zisk daného atributu vůči celému datasetu v uzlu N, jako kritérium vybereme atribut s největším ziskem.

$$Entropy(S) = \sum_{j=1}^{k} p_j \log_2 p_j$$

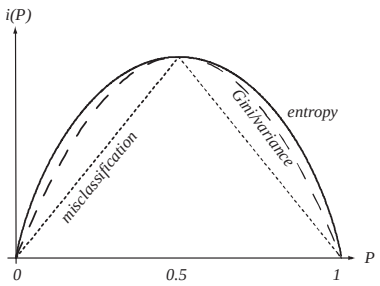
$$Gain(S, S_j) = Entropy(S) - Entropy(S, S_j)$$

$$např.\ Gain(PlayGolf, Outlook) = Entropy(PlayGolf) - Entropy(PlayGolf, Outlook)$$

- **Gini index** - měří divergenci mezi pravděpodobnostním rozdělením hodnoty cílového atributu.

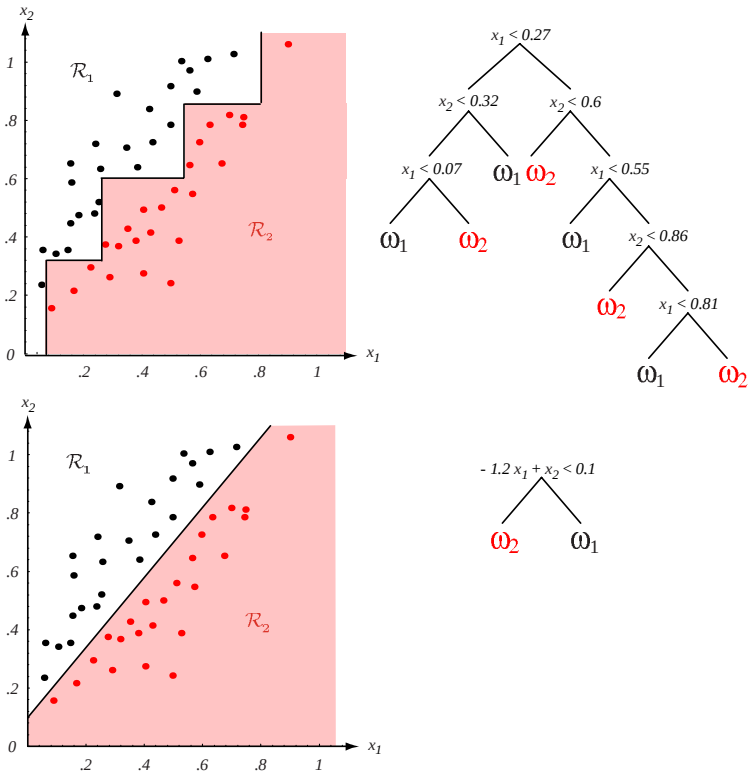
$$Gini(S) = 1 - \sum_{j=1}^{k} p_j^2$$

Každý algoritmus využívá jiná kritéria, nejznámější ID3 a C4.5 jsou založena na Entropii, zatímco CART je založen na Gini indexu. Krom výše uvedených existuje celá řada dalších technik: Likelihood-Ratio Chi–Squared Statistics, DKM Criterion, Normalized Impurity Based Criteria, Gain Ratio, Distance Measure, Binary Criteria, Twoing Criterion, Orthogonal (ORT) Criterion, Kolmogorov–Smirnov Criterion... více v knize [15, kapitola 9].



Obrázek 18: Binární klasifikace - Entropie, Gini a Error rate, zdroj [14]

Na obrázku 19 je názorně předveden vliv split kritéria na kvalitu a složitost výsledného modelu. Multivariate split se správnou podmínkou dokáže lépe rozdělit množinu při větší generalizaci a druhý strom tudíž bude schopen přesněji klasifikovat nové hodnoty.



Obrázek 19: Split kritérium a kvalita modelu, zdroj [14]

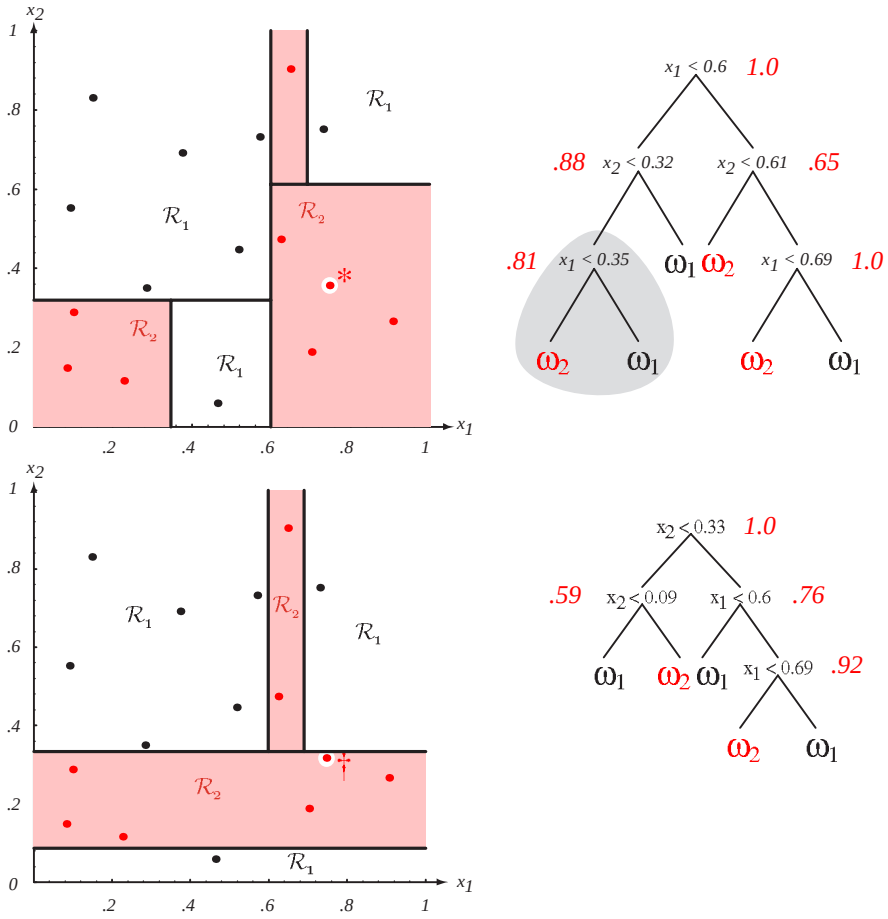
6.2.1.3 Pruning - prořezávání

K větvení stromu by bez jakéhokoliv *stop* kritéria mělo dojít až při 100% pokrytí trénovací množiny. To by však mohlo vést k nechtěnému přeučení a krom zbytečně složitého stromu by výsledný model nebyl schopen generalizace na nová data. Abychom udrželi model dostatečně přesný, ale přitom schopný generalizace, můžeme využít tyto dvě techniky, které dovedou zobecnit generovaný strom a potlačit možný šum v trénovacích datech:

- **Pre-pruning** (stop kritéria) - při sestavování stromu detekujeme hranici, kdy zastavíme další větvení. Ta může být krom základního případu, kdy všechny instance spadají do jedné třídy, daná maximální hloubkou stromu, stanovením minimálního množství vzorků v koncovém uzlu (listu), poměrem vzorků rozdělených split kritériem do dalších větví, nebo toleranční hranicí *best-split* kritéria, anglicky threshold, viz [15].
- **Post-pruning** (zpětné prořezávání) - necháme sestavit 100% model a následně zpětně odebíráme pravidla, jejichž odstranění nepřekročí tolerovanou chybu.

6.2.1.4 Shrnutí

Rozhodovací stromy jsou vhodné klasifikátory pro řešený problém, neboť jsou schopny s dobrou přesností řešit problém i přes chybějící data. Základní rozhodovací stromy jsou velmi citlivé na vyváženost trénovací množiny. Jak je vidět na obrázku 20, vypuštění jediné hodnoty z trénovací množiny, může vést k sestavení zcela jiného stromu. Z tohoto důvodu budou využity spíše jako kontrolní mechanismy díky dobré vizualizaci modelu při jeho tvorbě a hledání vhodných normalizací, pro reálné nasazení bude použita jedna z variant, které si představíme v dalších kapitolách.

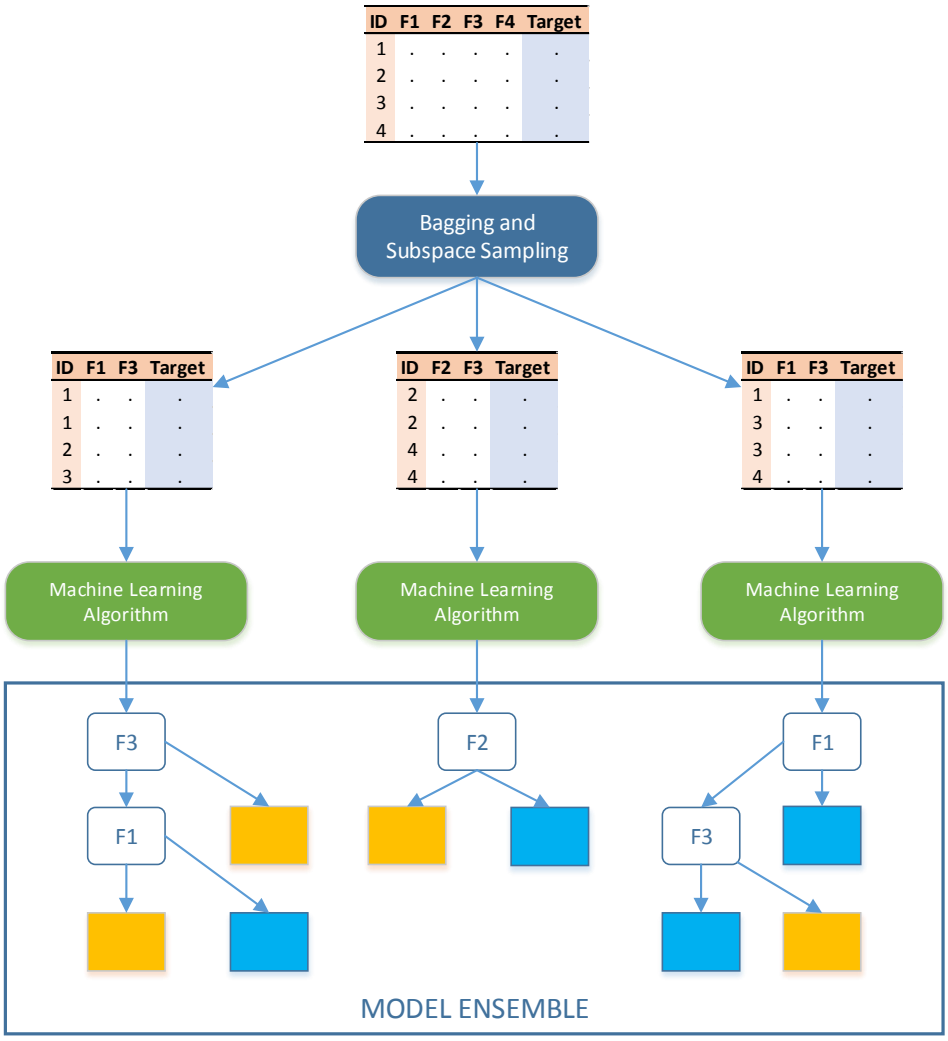


Obrázek 20: Citlivost rozhodovacího stromu na trénovací množinu, zdroj [14]

Výpočetní náročnost konstrukce stromu je $\mathcal{O}(d \cdot n^2 \cdot \log(n))$ viz [14], kde d je dimenze (počet atributů) a n je počet vzorků trénovací množiny. ML knihovny mívají optimalizace založené na předtřídění vzorků a udržování průběžných statistik, díky čemuž dovedou tento problém řešit s nižší složitostí, obvykle $\mathcal{O}(d \cdot n \cdot \log(n))$.

6.2.2 Random Forest

Je založen na obecné technice zvané *Bagging*. Ta spočívá v rozdělení trénovací množiny na několik podmnožin náhodným výběrem instancí (viz obr. 21), kdy stejná instance může být v podmnožině obsažena více než jednou.



Obrázek 21: Bagging - generování podmnožin z trénovací množiny, zdroj [14]

Pro každou podmnožinu je následně vygenerován samostatný rozhodovací strom. Výsledná třída je vybrána hlasováním z množiny výsledků dané jednotlivými stromy. Tato základní myšlenka byla v různých implementacích dále vylepšena o náhodný výběr atributů pro rozhodování nebo randomizaci při výběru split kritéria.

Výhodou tohoto přístupu z pohledu škálovatelnosti je snadná paralelizace výpočtu, neboť každý strom počítáme nad samostatnou podmnožinou a výpočet pak nevyžaduje krom finálních výsledků synchronizaci, což je důležité především u rozsáhlých datasetů.

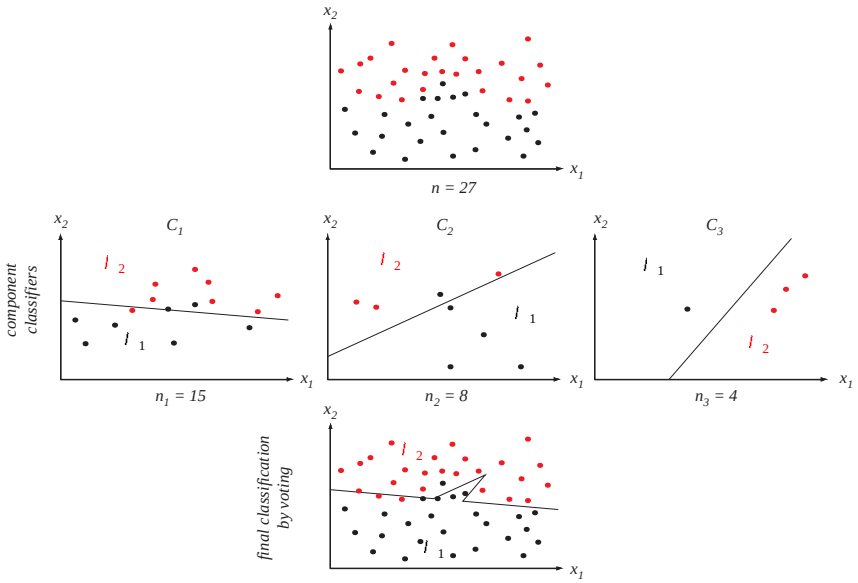
6.2.3 Boosted Decision Trees

Další varianta je založena na obecné technice zvané *Boosting*. Ta vychází z myšlenky, že skupina jednodušších klasifikátorů (*weak learner*) dovede ve výsledku vytvořit silný klasifikátor (*strong learner*). Jedná se o iterativní metodu, která obecně vytváří sadu klasifikátorů postupným přidáváním nových klasifikátorů tak, že jejich trénovací množinu tvoří vzorky, které byly již existujícími instancemi klasifikátorů chybně predikovány.

Nejvíce používaný algoritmus je AdaBoost, který začíná na náhodném výběru podmnožiny z trénovacích dat, jejichž váhy nastaví na $w[i] = \frac{1}{n}$, kde n je počet vzorků. Dle [13] pak v každém kroku:

1. Sestaví model pomocí váženého datasetu a spočte celkovou chybu predikce ε z jednotlivých chybných klasifikací.
2. Zvýší váhy chybně klasifikovaných instancí pomocí vzorce $w[i] \leftarrow w[i] \cdot \left(\frac{1}{2 \cdot \varepsilon}\right)$ a sníží váhy u korektně klasifikovaných vzorků dle $w[i] \leftarrow w[i] \cdot \left(\frac{1}{2 \cdot (1-\varepsilon)}\right)$.
3. Spočítá faktor spolehlivosti α tak, že se snižující se chybou ε se zvyšuje spolehlivost α dle vztahu $\alpha = \frac{1}{2} \cdot \log\left(\frac{1-\varepsilon}{\varepsilon}\right)$.

Výpočet je omezen maximálním počtem klasifikátorů a učících cyklů. Celková predikce je výsledkem majority vážených klasifikací jednotlivých klasifikátorů, viz obrázek 22. Na rozdíl od Random Forest je tento postup více citlivý na šum v trénovací množině. Algoritmus je odolný vůči přeučení, tzv. *overfitting*.



Obrázek 22: Boosting - původní dataset, tři slabé klasifikátory a výsledná silná klasifikace, zdroj [14]

6.2.4 Support Vector Machines

SVM (známý jako *Maximum Margin Classifier*) je v základu binární klasifikátor (rozšiřitelný na multi-class problém), jehož princip spočívá v hledání způsobu, jak lineárně oddělit dvě množiny tak, aby hranice vymezená dělící přímkou mezi instancemi tříd byla co nejširší. K nalezení dělící přímky využívá malé množství instancí trénovací množiny na hranici oblastí daných tříd, tzv. *support vectors*. Ke splnění lineární separability u nelineárních datasetů obvykle transformuje problém do mnohem vyšší dimenze, než je dimenze atributů trénovacích dat. Na obrázku 23 je vidět nejvhodnější rozdělení dvou tříd se dvěma atributy, hranice mezi dělící přímkou je co nejširší a rovnoměrně vzdálená od třech hraničních podpůrných vektorů.

Algoritmus je složen ze dvou základních částí, první řeší optimální rozdělení lineárně separabilního datasetu, druhá řeší transformaci nelineárně separabilního datasetu do takové dimenze, kde již jsou data lineárně separabilní.

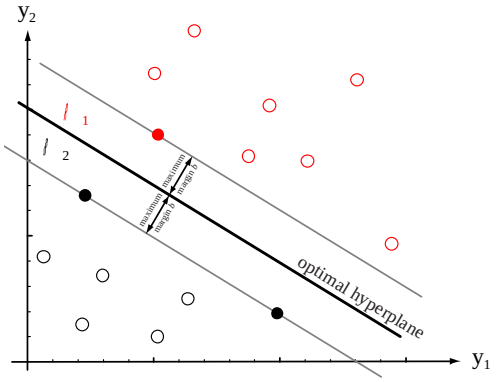
První část - lineárně plně separabilní SVM:

1. Prvky z trénovací množiny označíme jako dvojice $(\vec{x_i}, y_i), y_i \in \{-1, +1\}$, kde $\vec{x_i}$ je vektor z atributů i-tého prvku a y_i je výstupní třída ohodnocena číslem ± 1, podle toho, do které třídy daná instance patří.

2. Na základě předpokladu, že jsou data plně lineárně separabilní, platí že $\vec{w} \cdot \vec{x}_i + b \geq +1$ pro $y_i = +1$ a $\vec{w} \cdot \vec{x}_i + b \leq -1$ pro $y_i = +1$

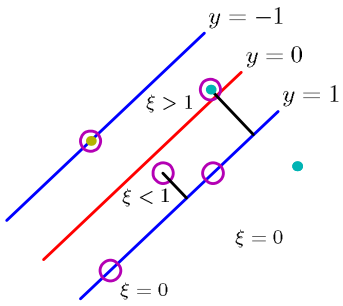
3. Hledáme nadrovinu danou rovnicí $\vec{w} \cdot \vec{x} + b = 0$, kde $\vec{w}$ je normálový vektor nadroviny a b je skalár, známý jako *bias*. Vektor $\vec{w}$ řídí natočení nadroviny a bias b reguluje šířku hranice, přesněji vzdálenost od okrajových *support vectors*.

4. Cílem optimalizační úlohy je nalézt maximální šířku hranice $\frac{2}{\|\vec{w}\|}$, v praxi se řeší jako minimalizační úloha pro $\frac{\|\vec{w}\|^2}{2}$, metodou Lagrange undetermined multipliers, viz [14] nebo [12].



Obrázek 23: SVM - optimální nadrovina (tři podpůrné vektory), zdroj [14]

V případě, že jsou data zatížena šumem nebo nejsou z jiného důvodu plně lineárně separabilní, tak se do výše uvedených vztahů zavede toleranční chyba ξ, tzv. *Soft Margin* a platí že, $\vec{w}\cdot\vec{x}_i+b \geq +1-\xi_i$ pro $y_i = +1$ a $\vec{w}\cdot\vec{x}_i + b \leq -1+\xi_i$ pro $y_i = +1, \forall i : \xi_i \geq 0$. Vliv na další výpočty a konkrétní postup lze nalézt v [12, kapitola 10.6.2]. Na obr. 24 jsou instance s $\xi_i > 1$ chybně klasifikované (tj. leží na špatné straně hranice), $0 < \xi_i < 1$ leží uvnitř toleranční hranice na správné straně a $\xi_i = 0$ jsou klasifikovány korektně.

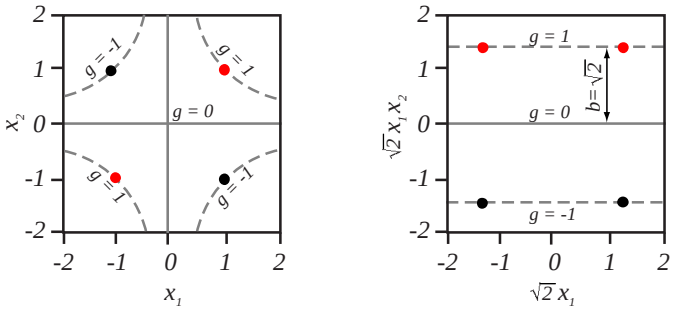


Obrázek 24: SVM - soft margins, zdroj [16]

Druhá část - nelineárně separabilní SVM: V tomto případě se provádí tzv. *Kernel trick*, při kterém přemapujeme problém z prostoru, ve kterém nejsou data lineárně separabilní, do vyššího prostoru, kde již dovedeme nalézt nadrovinu, která lineárně oddělí dané třídy. Samotné výpočty jsou prováděny v původním prostoru. Nejčastěji používané kernel funkce jsou:

- Linear - $kernel(u, v) = u^T v$
- Polynomial - $kernel(u, v) = \gamma(u^T v + c_0)^d$, parametry: γ, d, c_0
- Radial basis - $kernel(u, v) = exp\{-\gamma|u - v|^2\}$, parametry: γ
- Sigmoid - $kernel(u, v) = tanh\{\gamma u^T v + c_0\}$, parametry: γ, c_0

Příklad transformace problému na lineárně separabilní je vidět na obrázku 25 (funkce XOR). Zde bylo nutné 2D problém (x_1, x_2) transformovat do šesti-rozměrného prostoru $(1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$, ve kterém již bylo možné provést lineární oddělení.

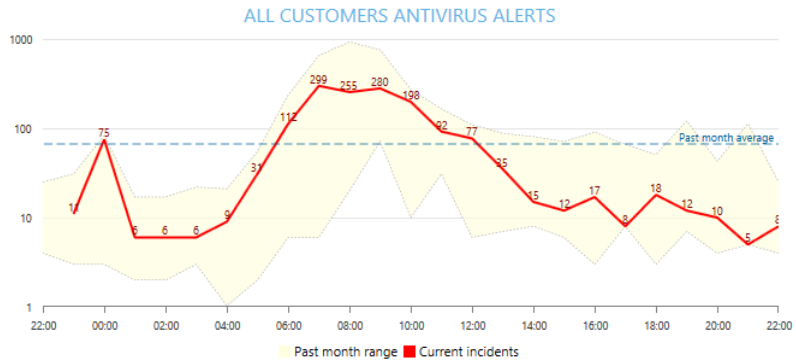


Obrázek 25: SVM - funkce XOR, 2D → 6D, (2D projekce), zdroj [14]

6.2.5 Modelování a predikce časových řad

Pro detekci outbreaku bude nutné využít kombinaci více technik. Základem je stejně jako ve všech podobných programech tzv. *Baseline* (reálná ukázka antivirových dat z formuláře Virusmon, viz obr. 26[19]), tj. naučíme se z historických dat průměrné hodnoty, minima a maxima v čase, popř. další statistické ukazatele. Při překročení baseline můžeme očekávat nestandardní situaci. Zadání ale předpokládá, že systém bude schopen předpovědět toto překročení dříve, než k němu reálně nastane. Při modelování časově orientovaných dat (dle [20], rozlišujeme mezi dvěma základními průběhy:

1. Stacionární - střední hodnota ani rozptyl nejsou funkcí v čase, data se periodicky opakují, střední hodnota lze vykreslit konstantní funkcí. Např. funkce sinus.
2. Nestacionární - opak stacionární série, data mohou růst nebo klesat v čase, např. vývoj populace nebo mohou mít proměnlivou periodicitu.



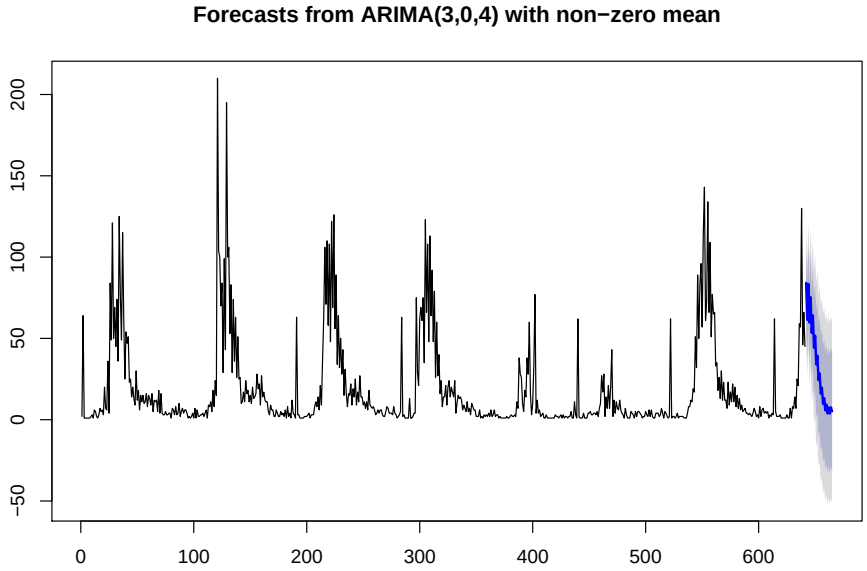
Obrázek 26: Projekce aktuálního průběhu levelu alertů vůči Baseline za 24 hodin

Pro predikci využijeme algoritmus ARIMA (Autoregressive integrated moving average). Základem algoritmu je:

- **AR** - Autoregression: regresní model - založen na předpokladu, že hodnota následujícího členu je výhradně závislá na hodnotě přímého předchůdce nebo několika předchozích pozorováních (vzorcích).
- **I** - Integrated: hledá rozdíly ve vzorcích s cílem izolovat periodické části k vytvoření stacionární série.
- **MA** - Moving-average: hledá závislosti mezi pozorovanými daty a reziduální chybou v předchozích vzorcích, např. stav skladu - náhlé vyprodání zásob.

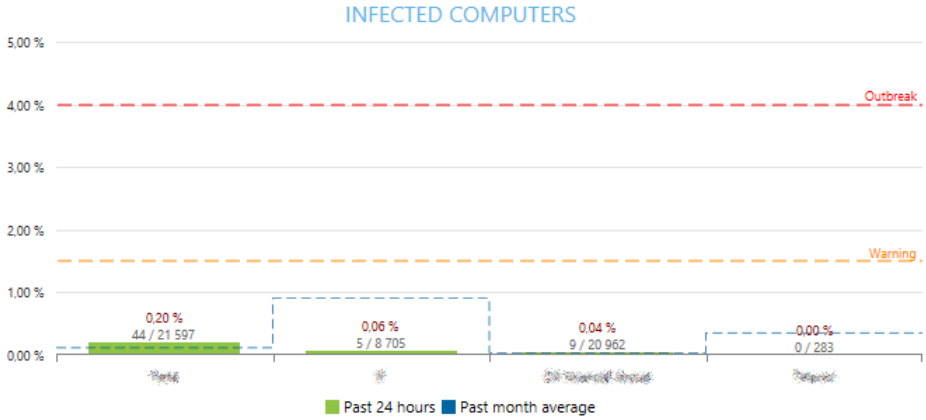
[19] Graf s baseline je inspirován systémem Event Management kolegy D. Skowronka.
[20] A Complete Tutorial on Time Series Modeling in R: `https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/`

Základem pro použití algoritmu ARIMA je stacionární série. Ta lze ověřit několika testy (základem je test homoscedasticity) a v případě, že data nejsou stacionární, museli bychom provést patřičné transformace. R obsahuje balík `Forecast` (stejně tak Python aj. mají k dispozici obdobné komponenty), který obsahuje funkci `auto.arima()`. Ta vyhledá v datech vzory včetně nestacionárních modelů a vrátí nejvhodnější ARIMA model s doporučenými parametry. Na obrázku 27 je vidět průběh zavirování za poslední týden. Modře je zobrazena predikce průběhu pomocí algoritmu ARIMA.



Obrázek 27: Predikce zavirování prostředí dle trendu z historických dat (týdenní data)

Krom časového průběhu je podle skupin zákazníků sledována celková infikace sítě, jako normální hodnota je stanoven průměr za poslední měsíc. Operátor v dohledovém centru pak může zareagovat na nestandardní situaci a provést ruční analýzu spolu s korektivními akcemi.



Obrázek 28: Zavirování celého prostředí (vybrány náhodné skupiny)

6.2.6 Microsoft R

R je programovací jazyk pro statistické výpočty a vizualizaci dat, který je distribuován se svým běhovým prostředím jako open-source platforma. Podobně jako Python těží z řady rozšiřujících balíčků, které přidávají další funkce nebo alternativní implementace k základní syntaxi jazyka. Těch je v dnešní době přes deset tisíc a díky nim je velmi dobře použitelný pro analýzu dat i strojové učení.

Nevýhodou standardního běhového prostředí R je nutnost umístit všechny objekty v paměti RAM. Spolu s nízkým výkonem nativních R funkcí a nemožností škálování výpočtů na více serverů je zpracování většího množství dat ve standardním R pro některé případy nepoužitelné a nemusí jít pouze o Big Data problémy. Tato omezení spolu s rozšířením o Big Data platformu vyřešili ve firmě Revolution Analytics přepsáním vybraných implementací do nativního jazyka C nad Intel MKL[21] knihovnou, datovou strukturou XDF, která umožní ukládat objekty na disk a škálovat výpočty přes Hadoop aj. pomocí ScaleR, podporou paralelních výpočtů, GPU a dalších prvků. Toto řešení bylo nabízeno jako samostatné běhové prostředí ve třech edicích, od open-source až po komerční R Server.

V roce 2015 zakoupil Microsoft tuto společnost a jejich komerční platformu R Server mimo jiné integroval přímo do SQL Serveru 2016 jako službu SQL Server R Services. Integrace umožňuje v rámci SQL dotazu volat R script spolu s obousměrnou komunikací, bez nutnosti exportu a importu dat přes externí soubor, s plnou podporou komerčních funkcí. Standardně je script vyvíjen lokálně nad prostředím Microsoft Open R client, který je např. omezen na paralelizaci v max. dvou vláknech, umístění objektů je možné pouze v RAM apod. Až při produkčním nasazení na R Server se využívají plné možnosti škálovatelnosti a dalších prvků dostupných pouze v komerční verzi. Více informací o této platformě viz [11], v současnosti již nabízí i Microsoft implementace známých algoritmů v balíčku MicrosoftML, HPC clustery, škálování přes Azure R servery, HDInsight a další Microsoft technologie, přičemž je zde zachována podpora i pro Hadoop, Linux či Teradata. Všechny funkce s prefixem `rx` např. `rxDForest` jsou napsány v technologii `ScaleR`[22] a podporují škálovatelnost od paralelních výpočtů a zpracování dat na disku až po distribuované systémy.

Tato práce využívá i balík MicrosoftML, který není dostupný v aktuální verzi SQL Server 2016 SP1, je součástí SQL 2017 CP2+. Pro aktualizaci R prostředí v SQL 2016 je možné využít `sqlBindR.exe`, který je dostupný po instalaci nejnovějšího R Server.

[21] Intel Math Kernel Library - vysoce optimalizovaná knihovna s matematickými funkcemi pro Intel procesory s podporou paralelizace, viz `https://software.intel.com/en-us/intel-mkl`.

[22] ScaleR `https://msdn.microsoft.com/en-us/microsoft-r/scaler/scaler`

6.3 Vytvoření gramatiky pro pravidlové systémy pomocí ANTLR

Moduly pro sestavení instrukcí, vyhledání kontaktů a výstupní akce jsou založeny na systému prioritních pravidel, které jsou generalizovány od nejvíce konkrétních (mají nejvyšší prioritu), až po obecná. Evaluace je ukončena prvním platným pravidlem. Od systému pro evaluaci pravidel očekáváme, že bude obecný, znovupoužitelný pro libovolný modul a přitom umožní popsat jakýkoliv z aktuálně známých případů. Budeme vycházet z následujících požadavků:

- Pravidlo je schopné evaluovat jakýkoliv model, aby šlo dané moduly využít i pro další problémové domény jako je SIEM, IPS, interní procesy apod.
- Pravidla jsou dostatečně jednoduchá pro administrátora.
- Podporují neomezené zanoření závorek, negaci, spojení podmínek pomocí AND, OR, srovnávací operátory a kvantifikátory pro případ podmínek ze složených kolekcí.
- Priorita operátorů musí být platná vůči logickým pravidlům: Závorka > AND > OR...
- V případě potřeby můžeme přiřadit vlastní funkce, matematické operace apod.
- Pro snadnější správu pravidel musí umět systém *syntax highlighting*, neboli barevné zvýraznění kódu.

Nejjednodušší řešení je využít některý ze skriptovacích jazyků, dostupných pro C# (CScript, Python, LUA), které umožňují volat skripty z aplikace včetně obousměrného předání proměnných. Pravidla by však nebyla dostatečně jednoduchá, navíc bychom ztratili kontrolu nad tím, co pravidlo dělá (např. náročný databázový dotaz). Tím však skriptovací jazyky nezavrhujeme, s úspěchem využíváme nativní skriptování z *Microsoft.CodeAnalysis.CSharp.Scripting* pro přepisovací engine v modulu, který by vyžadoval příliš častou kompilaci a deployment. Není však součástí této práce, takže se tím nebudeme dále zabývat.

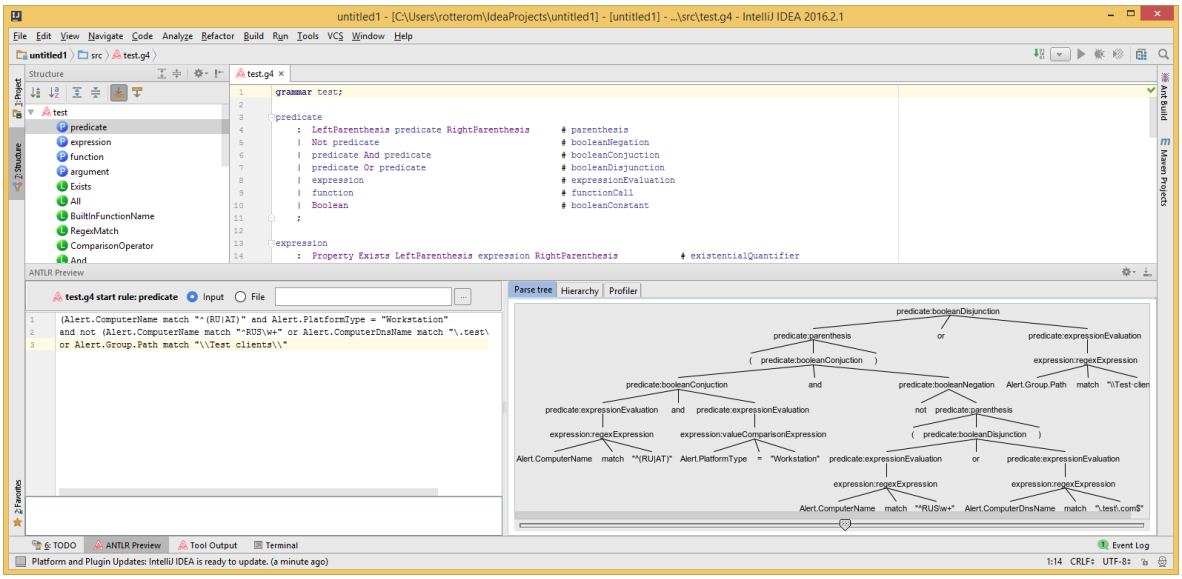
Na základě požadavků o prioritě a zanoření závorek je zřejmé, že budeme potřebovat bezkontextovou gramatiku. Zde máme opět několik možností, namísto psaní vlastního rekurzivního sestupu stromem můžeme využít některý z nástrojů pro definici gramatik, které umí i vygenerovat kostru tříd pro parsování a evaluaci generovaného jazyka. Nejvíce používané jsou ANTLR, Gold parser, YACC, Grammatica a řada jiných. Vzhledem k nulovým předchozím zkušenostem s praktickým návrhem gramatik byl zvolen ANTLR, který má velmi dobrou podporu při návrhu, neboť si můžeme gramatiku rovnou testovat i vizualizovat, aktuálně již i nativní podporu pro C# a Visual Studio a také skvěle zpracovanou příručku [18], která obsahuje jak teorii, tak příklady. ANTLR podporuje $LL1$ gramatiku, což však pro naše zadání plně dostačuje, striktnost a determinismus jsou vzhledem ke zkušenostem vlastně i výhodou oproti $LALR$.

ANTLR řeší jak lexer, tak parser. Lexikální tokeny začínají velkým písmenem, viz výpis 7, u složitějších gramatik jsou definovány v samostatném souboru. Plná kontrola nad jazykem umožňuje mimo jiné zavést uživatelsky přívětivé konstrukce jako `LastReaction()> 7 days`, kde si časový interval vypočteme z čísla a konstanty (den, měsíc, rok...), namísto abychom uživatele nutili převádět všechny varianty na hodiny. ANTLR nám pak rovnou dodá hodnotu číselné části a textové konstanty. Princip je založen na konvencích, stačí označit návěští gramatik a lexer tokeny. Ve vygenerovaných třídách pak např. na základě návěští `# valueComparisonExpression` nalezneme metodu `VisitValueComparisonExpression(context)` a v rámci ní máme díky označení `val=(Boolean | Integer...)` přístup k enumu `context.val.Type`, který nabývá hodnot dle gramatiky, např. `VqplLexer.Boolean`.

```
grammar Vqpl;

predicate
    : LeftParenthesis predicate RightParenthesis # parenthesis
    | Not predicate                              # booleanNegation
    | predicate And predicate                    # booleanConjuction
    | predicate Or predicate                     # booleanDisjunction
    | expression                                 # expressionEvaluation
    | function                                   # functionCall
    | Boolean                                    # booleanConstant
    ;

expression
    : Property Exists LeftParenthesis expression RightParenthesis           # existentialQuantifier
    | Property All LeftParenthesis expression RightParenthesis              # universalQuantifier
    | Property RegexMatch String                                            # regexExpression
    | Property ComparisonOperator val=(Boolean | Integer | Double | String) # valueComparisonExpression
    | function ComparisonOperator val=(Boolean | Integer | Double | String) # functionComparisonExpression
    ;

function
    : BuiltInFunctionName ((LeftParenthesis argument (',' (argument))* RightParenthesis) | (LeftParenthesis RightParenthesis))
    ;

argument
    : Property
    | String
    | Integer
    ;

Exists                : 'exists';
All                   : 'all';
BuiltInFunctionName   : 'AlertsInPastDays' ;
RegexMatch            : 'match' ;
ComparisonOperator    : '=' | '==' | '<=' | '<' | '>' | '>=' | '!=' | '<>';
And                   : 'and' ;
Or                    : 'or' ;
Not                   : 'not' ;
Boolean               : 'true' | 'false' ;
Property              : Identifier (Dot Identifier)* ;
Identifier            : [a-zA-Z_] [a-zA-Z_0-9]* ;
Integer               : Digit+ ;
Double                : Digit+ Dot Digit+ ;
Digit                 : [0-9] ;
Dot                   : '.' ;
LeftParenthesis       : '(' ;
RightParenthesis      : ')' ;
Comma                 : ',' ;
String                : UnterminatedString '"' ;
Whitespace            : [ \r\n\t] + -> skip ;
LineComment           : '//' ~[\r\n]* -> skip ;
UnterminatedString    : '"' (~["\\\r\n] | '\\' (. | EOF))* ;
```

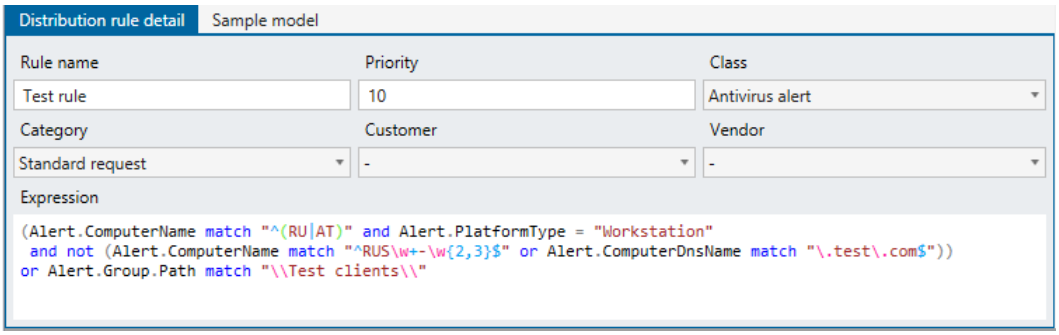
Výpis 7: ANTLR - návrh gramatiky pro pravidlové systémy

Pro návrh gramatiky existuje několik variant editorů, buď ve formě samostatného ANTLRWorks nebo doplňků pro známé IDE jako je IDEA, Eclipse, Netbeans. Nejzdařilejší IDE se jeví doplněk pro IntelliJ IDEA, viz obrázek 29, který v reálném čase kontroluje syntaxi, v případě chyby řekne přesně, ve kterém místě nastala a zobrazí důsledek červeně v syntaktickém stromu. Zároveň kontroluje i levou rekurzi a další chyby v návrhu LL1 gramatiky.

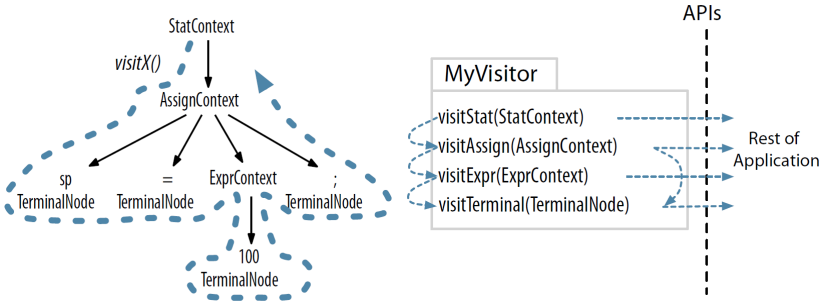


Obrázek 29: GUI pro návrh ANTLR gramatiky dostupné jako doplněk pro IntelliJ IDEA

Zvýraznění syntaxe je v GUI aplikaci vyřešeno pomocí knihovny `AvalonEdit`, která umožňuje definici syntaxe vlastního jazyka. Návrh je inspirován zvýrazněním ve Visual Studio, zvýraznění regex hodnot pak vychází z pluginu Resharper. Výsledná kombinace je vidět na obrázku 30. Správné zvýraznění pomáhá snížit chybovost, především u regex hodnot, které jsou v některých našich případech poměrně rozsáhlé. (pozn. regulární výrazy jsou v administračním prostředí využívány skoro všude, takže se očekává, že operátor `match` bude nejvíce využívaný, z toho důvodu je kladen důraz na zvýraznění regex syntaxe).



Obrázek 30: Zvýraznění syntaxe v GUI aplikaci



Obrázek 31: Třídy generované z gramatiky dle návrhového vzoru Visitor, zdroj [18]

ANTLR generuje C# třídy dle návrhového vzoru *Visitor*, viz obrázek 31. Při implementaci pak dědíme z vygenerované bázové třídy a přepisujeme ty metody, které potřebujeme upravit. V aplikaci je reflexe využita na mnoha místech, proto jsou metody usnadňující práci pomocí reflexe nabídnuty jako služba přes `IApplicationReflectionService`. Aby bylo možné se navigovat přes kompozitní objekty, tak si třída udržuje zásobník objektů, v případě zanoření se tento objekt umístí na vrchol zásobníku, po evaluaci se z něj odstraní a pokračuje se ve vyhodnocování původních pravidel. Např. evaluace *Alert.InfectedFiles exists (Name match "test")*, kdy je modelem pro evaluaci v závorce kolekce InfectedFiles, viz výpis 8, *VisitExistentialQuantifier*. V rámci aplikace je za evaluaci zodpovědná služba `IVqplParsingService`.

```
public sealed class VqplVisitor : VqplBaseVisitor<bool>
{
   private readonly Stack<object> _models;
   private readonly IApplicationReflectionService _reflectionService;

   public VqplVisitor(object model, IApplicationReflectionService reflectionService)
   {
      _models = new Stack<object>();
      _models.Push(model);
      _reflectionService = reflectionService;
   }

   public override bool VisitRegexExpression(VqplParser.RegexExpressionContext context)
   {
      var property = _reflectionService.GetPropertyValue(_models.Peek(), context.Property().GetText()) ?? string.Empty;
      var pattern = context.String().GetText().RemoveQuotes();
      var result = Regex.IsMatch(property.ToString(), pattern, RegexOptions.IgnoreCase | RegexOptions.Singleline);
      return result;
   }

   public override bool VisitExistentialQuantifier(VqplParser.ExistentialQuantifierContext context)
   {
      var collection = _reflectionService.GetPropertyValue(_models.Peek(), context.Property().GetText());
      foreach (var entry in (IEnumerable)collection)
      {
         _models.Push(entry);
         var result = Visit(context.expression());
         _models.Pop();
         if (result) return true;
      }
      return false;
   }

   public override bool VisitBooleanNegation(VqplParser.BooleanNegationContext context)
   {
      return !Visit(context.predicate());
   }...
```

Výpis 8: Ukázka implementace VQPL visitor

6.4 Dynamické vytvoření instrukcí

Tvorba instrukcí nebo obecně libovolného výstupního textu je v rámci celé aplikace jednotná, zajištěna službou `ITextTemplateSelector`, která nalezne nejvhodnější instrukce pro vybraný datový model, proces, a následně vyrenderuje přes engine, pro který je vybraná šablona napsána. Celý postup je zachycen na obrázku 32.

Obrázek 32: Proces sestavení instrukcí

V první fázi se na základě typu objektu (model) a požadované akci (např. standard request, sample request) pomocí pravidlového systému pro výběr instrukcí vyhledá nejvhodnější řešení. Pravidla jsou popsána jazykem VQPL, viz předchozí kapitola, a obsahují pouze podmínky a odkaz na reálnou šablonu. To umožní použít stejnou šablonu vícekrát. Následuje evaluace dynamických tagů. Ty nejsou nic jiného, než další šablony nebo statické texty spolu s podmínkou, pro který případ platí (pokud chceme tag platný vždy, pak nastavíme podmínku na true, např. podpis týmu). Platné tagy vyrenderujeme a přidáme do modelu. Ty, které nevyhoví podmínkám, budou při renderování odstraněny. Rozdělení šablony na hlavní a dynamické tagy je vhodnější varianta, než se snažit zavést dynamiku do šablon přes inline podmínky (if PlatformType = "Server"then "instrukce"), protože udrží šablonu jednodušší a čitelnější, navíc je možné stejný tag využít ve více šablonách.

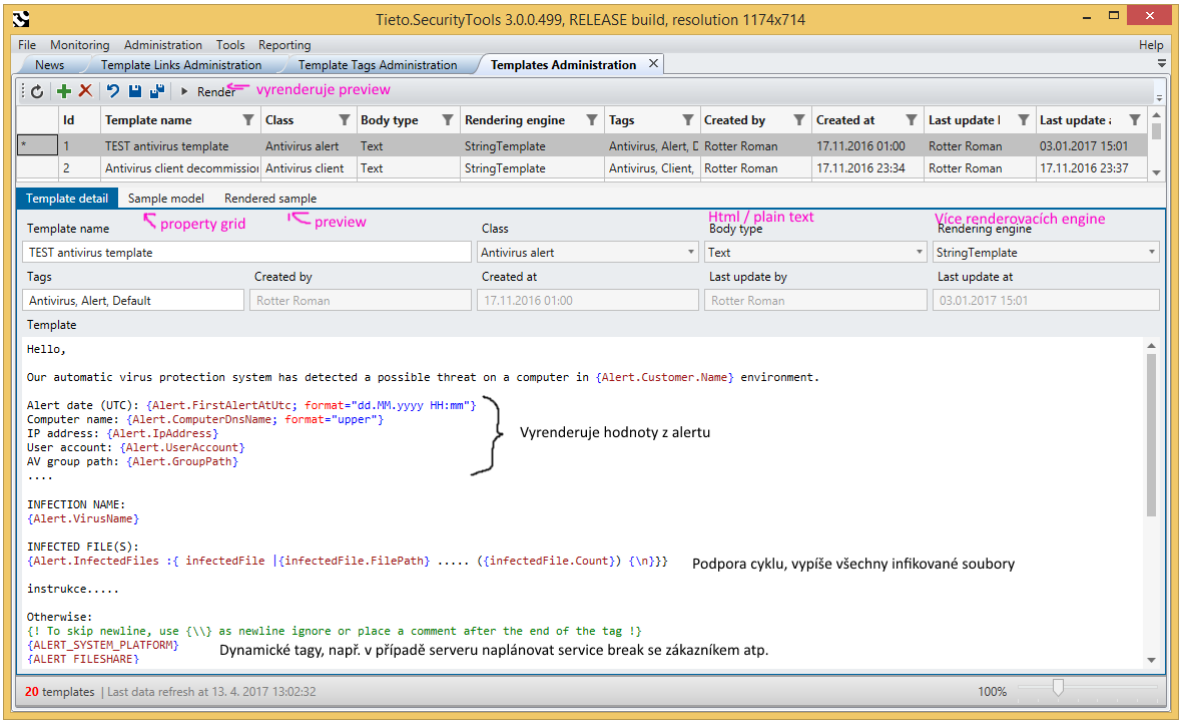
6.4.1 Template rendering engine

Systém byl navržen pro tři renderovací stroje, které se liší svými možnostmi a rychlostí. Hlavní engine je založen na `StringTemplate` knihovně, která nabízí vše co pro běžné šablony potřebujeme s dobou renderování: cold = 500ms, warm = 150ms. Pro renderování jednoduchých šablon `ReflectionRenderer`, který pomocí reflexe a operací s texty vyrenderuje šablonu maximálně v jednotkách milisekund, ovšem nepodporuje cykly, formátování a další prvky složitějšího engine. Jeho hlavní nasazení je renderování tagů, kde by při využití StringTemplate trvala celá operace několik vteřin. Třetí engine, který byl nakonec z produkční verze odstraněn, je Razor, který umožňuje i vlastní podmínky, makra apod. Renderování trvá: cold = 1800ms, warm = 750ms.

Zvýraznění syntaxe je opět řešeno vlastní definicí jazyka pro AvalonEdit, viz obr 33. Oproti obrázku je nyní ještě barevně odlišen dynamický tag od property. Původně byla zavedena pouze konvence Capital+Underscore, nyní lze okamžitě rozeznat, že daný tag neexistuje, protože známé tagy jsou dynamicky nahrány jako klíčové slova syntaxe.

V editoru si uživatel volí engine, zpravidla StringTemplate, a příznak zda tvoří HTML nebo PlainText šablonu, což se využívá např. pro výstup přes email.

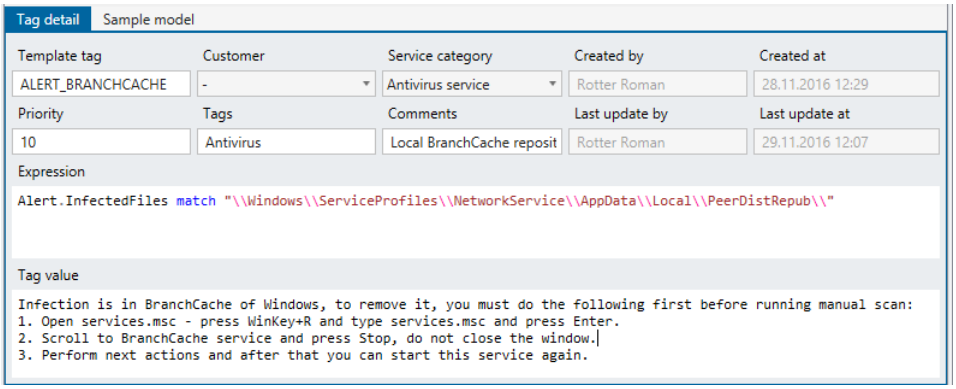
Pro psaní šablon je editor rozšířen o property grid, který se naplní podle datového typu (položka Class), pro který se píše šablona, např. Antivirový alert. Repository implementující `ISampleRepository<out T>` dovedou přes dependency injection poskytnout výchozí vzorek dané entity nebo vyhledat konkrétní entitu dle uživatelem zadaného ID. V property gridu pak má uživatel možnost upravit hodnoty dle potřeb a následně si nechat vyrenderovat editovanou šablonu. Případné chyby pak může rovnou opravit v rozpracované šabloně.



Obrázek 33: Editace šablon

6.4.2 Dynamické tagy

`TemplateTag` zajišťují minimální redundanci stejných instrukcí v šablonách a umožní s využitím malého množství šablon zasílat dostatečně konkrétní instrukce na většinu problémů. Mají vlastní editační formulář, tag se stejným identifikátorem může být definován vícekrát. Systém pak testuje podle priority shora dolů tagy se stejným kódem a použije první platný. To je vhodné např. pro odkaz na instalační balíček dle zákazníka, kdy v univerzální šabloně použijeme jediný tag a podle konkrétního zákazníka pak systém doplní vhodný odkaz. Na obrázku 34 je ukázka podmínky a statického textu, který se vyplní v šabloně místo tagu v případě, že podmínka platí. Tag může obsahovat renderování property z modelu, kde je využita stejná syntaxe jako v šablonách např. *{Alert.SignatureBasedDefinition.Version}*.

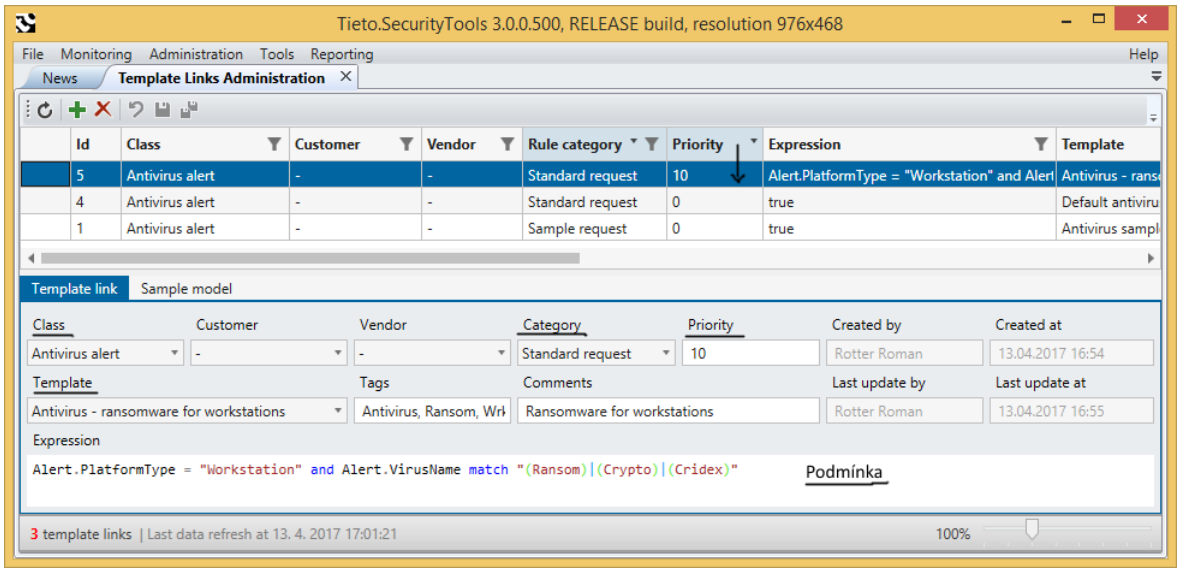


Obrázek 34: Editace dynamických tagů

6.4.3 Template selector

Je služba založená na systému pravidel, ze kterých vybere nejvhodnější šablonu pro řešení daného problému. Pravidlo se v systému nazývá `TemplateLink`, jeho jednoduchý příklad je vidět na obrázku 35, kde jsou definována dvě pravidla pro standardní instrukce a jedno pro vyžádání vzorku. Pokud je napadený klient stanice a malware obsahoval daný fragment, pak se použije vybrané pravidlo (10), které odkazuje na šablonu Ransomware workstation, jinak se pokračuje až na poslední (default), které má obvykle jedinou podmínku *true* a odkaz na Default template. Class definuje entitu nebo formulář, ke kterému se pravidlo vztahuje. Jedna datová entita se může využívat ve více formulářích, proto je spárováno tímto způsobem. Základní použití:

```
var instruction = _templateSelector.SelectAndRender(alert, ClassId.AntivirusAlert, Category.StandardRequest)
```



Obrázek 35: Definice pravidel pro Template selector

6.5 Eskalační engine

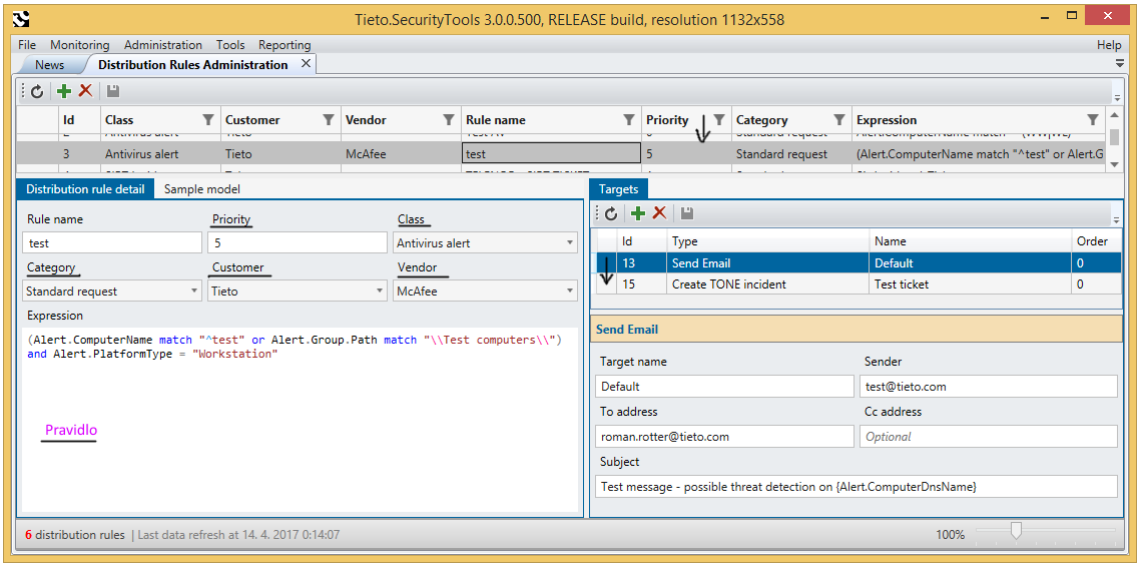
Tento modul redukuje množství pomocných dokumentů obsahujících informace, jakým způsobem a které osoby / týmy kontaktovat pro řešený problém. Zároveň také redukuje množství práce při vytváření daného požadavku, na pouhé zkontrolování připravených informací před odesláním, což je technicky nepovinná část, ovšem zatím chceme od systému konzervativní jednání.

Cílem modulu je zcela eliminovat pomocné Excel, Word a jiné dokumenty pro sdílení těchto informací, a zároveň také zrušit ruční přepisování nebo kopírování dat z jednoho rozhraní do druhého.

6.5.1 Distribution rule

Stejně jako v předchozím systému vycházíme ze souboru pravidel, vyhodnocených dle priority shora-dolů. V editoru (viz obr. 36) pouze upřesníme, pro jaký typ formuláře a akci se jedná (např. eskalace antivirového alertu, žádost o přeinstalování antiviru apod.). Obvykle zde řídíme jedno a více pravidel pro každého zákazníka zvlášť. Základní dvě pravidla obvykle rozlišují servery a stanice, které jsou spravovány různými týmy (datacentrum / lokální podpora). Můžeme však rozlišit i různá oddělení nebo divize společnosti.

Obecně pravidlem pouze určujeme, zda je pro daný problém platné a tudíž se použijí vztažené výstupní akce či nikoliv.

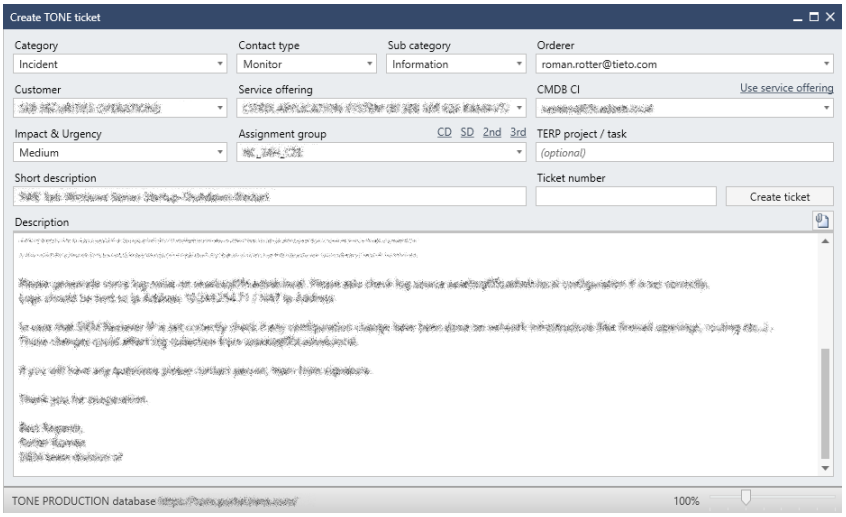


Obrázek 36: Správa distribučních pravidel

6.5.2 Distribution target

Každé pravidlo definuje jednu nebo více výstupní akcí, které reprezentují integraci s určitým systémem. Údaje pro vytvoření záznamu v externím systému, mohou být statické jako je příklad emailu na obrázku 36 nebo se mohou získat dynamicky. V některých firmách např. zodpovídá za počítač majitel a tak vyhledáme kontakt v Active Directory. U serverů máme obvykle v rámci CMDB nadefinovány konkrétní kontrakty a zodpovědné skupiny, takže si kontaktní údaje zjistíme na základě těchto informací.

Target je ekvivalentem výstupní akce a má za úkol připravit všechny potřebné kontaktní údaje, buď ze statických hodnot nebo vyhledáním v systému, obsahujícím potřebná data. Poté (volitelně) vyvolá dialog s připraveným formulářem, kde má specialista možnost změnit libovolný z údajů, upravit vygenerované instrukce nebo přidat přílohy.



Obrázek 37: Příklad výstupní akce

Programátor pak definuje, jestli vyvolá GUI dialog nebo odešle data do výstupního systému automatiky. Systém je navržen pro odesílání na více cílů, např. pro řešení nestandardních situací typu - manažer oddělení chce získávat kopie instrukcí z ITSM ticketu mailem apod. V přípravě je také podpora zpráv přes SMS. Uživatel v GUI nerozlišuje kam a jak chce instrukce poslat, existuje jen jedno tlačítko *Create ticket*, systém sám vyvolá správnou akci. Systém funguje i bez GUI, kdy je aktivní pouze M-VM vrstva (u Job serveru).

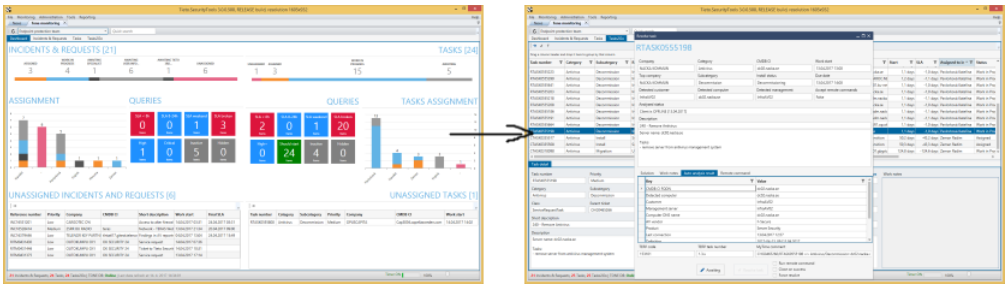
Implementace je řešena přes rozhraní `IDistributionTargetService` (tj. Target) se jmennou registrací do DI katalogu. Programátor pak využívá `IDistributionService`, která dle konfigurace aktivuje všechny výstupní moduly (Targets). Tímto způsobem bylo možné zcela oddělit konfigurační část a konkrétní pluginy, které implementují danou integraci, což je vhodné např. při přechodu na novější verzi API nebo jiný ITSM systém, kdy je nutné po dobu převodu zákazníků, např. rok, transparentně podporovat oba systémy.

6.6 False positive

Samostatná tabulka obsahující HASH souboru, název viru, výrobce a ukázku detekovaných souborů, která slouží pro automatické označení známých false positive import službou. Vždy se jedná o dočasný problém, kdy antivir chybně detekuje čistý soubor a končí opravou této chyby ze strany výrobce nebo se jedná o podezření na false positive a následné potvrzení reálné hrozby. Specialista si pomocí monitorovacího GUI vyžádá získání vzorku, součástí této akce je automatické založení záznamu do této tabulky. Po jeho získání provede analýzu a v případě chybné detekce vyžádá u výrobce odstranění z definic. Uživatelé mají k dispozici GUI s CRUD operacemi nad touto tabulkou, kde mohou aktualizovat stav daného false positive (under investigation, confirmed false positive, real infection).

6.7 Instalace / decommission klientů

Proces začíná vytvořením decommission úlohy v ITSM systému. Operátor zpracovává tyto požadavky pomocí monitorovacího GUI, kde systém detekuje, zda se jedná o Antivir install/decommmission požadavek. V případě instalace provede kontrolu v tabulce statusu klientů (assets), kde zjistí zda je daný antivir nainstalován, má aktuální definice, komunikuje atd. Pokud kontrola projde, vygeneruje report o aktuálním stavu a zavře s ním tiket, jinak vloží záznam do worklogu a přepne tiket do stavu awaiting. U decommission zkontroluje, zda daný server ještě existuje. Pokud je již smazán, navrhne zavření ticketu. V opačném případě si v tabulce, ve které se přenáší informace o management serverech zkontroluje, zda jejich RemoteAgent podporuje vzdálené příkazy (aktuálně pouze McAfee s agenty připojenými přes MessageBus). Systém vytvoří datovou entitu `RemoteCommnad`, který obsahuje název a GUID daného klienta, adresu cílového serveru, číslo tiketu a textově plná konstanta příkazu, např. `CommandText="Antivirus.Decommission"`. Command uloží do lokální databáze a zapíše informace do worklogu. SyncServices pak přes MessageBus obsahující frontu příkazů přenesou command až na vzdálený RemoteAgent, kde se pomocí DI a hodnoty CommandText aktivuje příslušný modul. Ten vykoná příkaz pomocí API management serveru a stejným způsobem vrátí odpověď. V případě úspěchu se navrhne zavření tiketu, jinak je nutné řešit problém ručně.



Obrázek 38: GUI podpora pro monitorování a řešení tasku

7 Testování a tvorba modelu pro klasifikaci alertů

V předchozím textu jsme se seznámili s podobou antivirového alertu, procesy a klasifikačními algoritmy. Nyní za pomoci těchto algoritmů a zpětné vazby z chybných klasifikací, budeme hledat výchozí model pro produkční nasazení, pomocí následujícího postupu:

1. Z množiny všech alertů si náhodně vybíráme alerty. Ty, které byly správně vyřešeny, označíme certifikačním razítkem a zároveň vyřadíme podobné alerty z dalšího hledání. Vzhledem k množství a lidské chybě je třeba předpokládat existenci špatných ohodnocení včetně protichůdných řešení. Začneme tedy s menší množinou, kterou bude snadnější zkontrolovat a postupně ji budeme rozšiřovat.

2. Alerty znormalizujeme dle kapitoly 7.1.

3. Trénovací množinu necháme naučit a následně klasifikovat vybranými algoritmy - např. SVM vyniká v přesnosti naučení celé množiny, čímž nalezneme anomálie, DT na vizuální kontrolu rozdělení trénovacích dat, aby nenastala situace, kdy model tvoříme např. jen ze smazaných alertů podobných vlastností. Chybně klasifikované instance ručně zkontrolujeme a hledáme příčinu chyby:

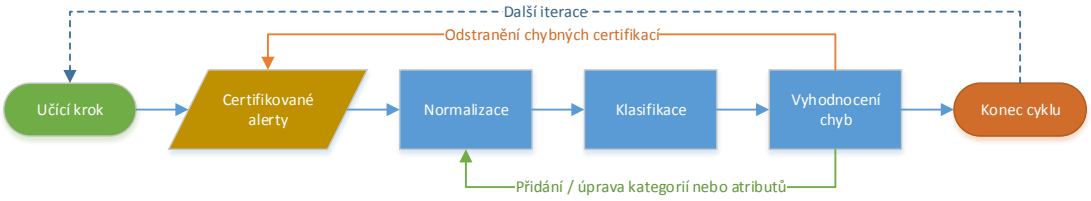
   (a) Chyba vzniklá chybějícím atributem nebo jeho kategorií - např. u kategorie atributu `FilePathCategory` nerozlišíme cestu v systému Windows, soubor v tomto adresáři bude mít cestu klasifikovánu jako `Others` a výsledná klasifikace nesplní očekávání. Stejně tak nedostaneme očekávané výsledky, pokud nerozlišíme např. stanice od serverů.

   (b) Chyba vzniklá špatným ohodnocením.

   (c) Algoritmus není schopen nalézt dostatečně kvalitní separační funkci.

4. V závislosti na chybě provedeme korektivní akce a přidáme další alerty do trénovací množiny. Tyto kroky opakujeme tak dlouho, až je trénovací množina dostatečně velká.

Celý postup lze graficky reprezentovat procesem na obrázku 39.

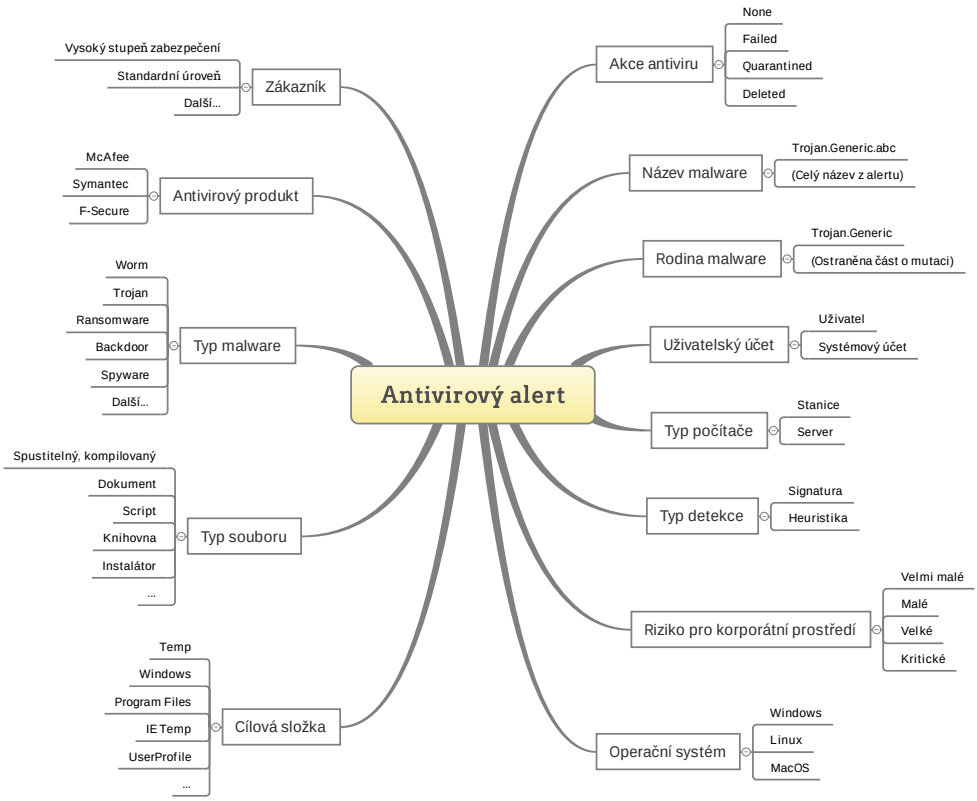


Obrázek 39: Postup při tvorbě modelu.

7.1 Normalizace alertu na základní atributy a jejich výběr

V analýze požadavků jsme si představili obecný formát antivirového alertu s atributy, které má většina antivirů (viz kapitola 4.5). V praxi jsou dostupné obvykle ještě informace o HASH souboru, procesu, digitálním podpisu, aplikaci, v případě síťového napadení i zdrojový počítač, u heuristické detekce zase rating reputace a další informace závislé na produktu a modulu, který daný malware detekoval. Pro klasifikaci však využijeme základní informace, dostupné pro všechny antivirové alerty.

Na myšlenkové mapě (viz obr. 40) jsou zachyceny klíčové atributy s ilustrací kategorií, které budeme zpřesňovat na základě testování definované v předchozí kapitole. Klasifikátor by měl rozlišovat např. mezi {exe, com, jar} vs {zip, doc, html} typy souborů, stejně tak detekci v adresářích, kde má právo zápisu pouze uživatel s administrátorským oprávněním. Dále mezi malware s vysokým rizikem šíření ve firemním prostředí a dopad jeho chování, např. pokusy o uhodnutí hesla v prostředí s Active Directory může zablokovat účty řadě uživatelů, stejně tak napadení jednoho počítače Ransomware v případě, že má namapováno síťové úložiště, může znehodnotit dokumenty celého oddělení.



Obrázek 40: Normalizace antivirového alertu - definice atributů

7.2 Srovnání klasifikačních algoritmů

Použité implementace algoritmů jsou již v základu navrženy tak, že si vypočtou obecně doporučené parametry podle dimenze a velikosti trénovacího datasetu. Ruční volbou parametrů pak můžeme klasifikaci zpřesnit nebo naopak zhoršit. Tím, že nevíme, jak jsou data v trénovací množině rozmístěna, tak budeme experimentálně hledat vhodné nastavení. Některé algoritmy jsou omezeny na počet kategorií / atribut (viz kapitola 7.2.5). Začneme s více normalizovanou množinou, která odpovídá přibližně výuce nových členů týmu a je použitelná pro všechny algoritmy. Tzn. kategorie malware, cílová složka, akce antiviru, typ souboru, ale neobsahuje např. detailní informace typu název malware. Nejprve je otestován vliv parametrů na přesnost[23]. Následně jsou algoritmy srovnány z pohledu výkonu a přesnosti jak u certifikované množiny, tak na velkém objemu zcela neznámých dat.

7.2.1 Support Vector Machines

Na algoritmus má vliv počet vzorků a atributů trénovací množiny, volba kernelu a dále parametry γ, který určuje, jak vzdálené body od dělící hranice budeme započítávat (krom lineárního jádra) a $C(cost)$, která nastavuje hladkost hranice, tj. jak detailně bude kopírovat hraniční vektory. c0 (platný pro polynomiální a sigmoid jádro) neměl vliv, takže je ponechán v defaultu (0).

Radial	Gamma				
Cost	0,01	0,1	0,3	0,7	1
1	87,6%	92,4%	91,6%	90,7%	88,0%
10	91,6%	92,9%	92,9%	91,1%	88,9%
100	93,8%	92,4%	92,9%	91,1%	88,9%
500	92,9%	92,4%	92,4%	91,1%	88,9%
1000	92,0%	92,4%	92,9%	91,1%	88,9%

Linear	
Cost	Accuracy
1	91,1%
10	93,8%
100	92,9%
500	92,9%
1000	92,9%

Polynomial	Gamma				
Cost	0,01	0,1	0,3	0,7	1
1	52,9%	92,0%	92,4%	92,4%	92,4%
10	52,9%	92,9%	92,4%	92,4%	92,4%
100	88,0%	92,4%	92,4%	92,4%	92,4%
500	89,8%	92,4%	92,4%	92,4%	92,4%
1000	92,0%	92,4%	92,4%	92,4%	92,4%

Sigmoid	Gamma				
Cost	0,01	0,1	0,3	0,7	1
1	86,2%	88,9%	68,4%	63,1%	49,8%
10	90,2%	92,9%	62,7%	59,6%	57,8%
100	91,1%	92,4%	61,8%	59,6%	57,8%
500	91,6%	92,4%	66,2%	59,6%	57,3%
1000	92,9%	92,4%	60,4%	59,6%	57,3%

Tabulka 2: Vliv parametrů SVM, certifikované data, poměr 70:30, 700 vzorků

Z tabulky 2 a článku autorů LIBSVM [17], lze předpokládat, že celý dataset je lineárně separabilní, anebo jsou dobré výsledky s *linear* kernelem dané aktuálním poměrem vzorků vůči počtu atributů. Druhý případ se v budoucnu může měnit spolu s rozšiřováním trénovací množiny a přidáváním kategorií. Průběžně tedy budeme kontrolovat scripty generující tyto tabulky, jestli je rozložení v množině aktuální nebo vyžaduje změnu parametrů.

[23] Nalezené hodnoty jsou platné pro aktuální dataset. Až bude množina v budoucnu rozšířena, tak je nutné provést kontrolu a případnou korekci parametrů dle aktuálního souboru dat.

Při testu detekujícím anomálie (tj. trénovací množina = testovací) využíváme *Radial* kernel ($cost = 100, \gamma = 0.1$), který je schopen velmi přesně definovat hranice mezi třídami a dosahuje 98% přesnosti. Chyby jsou z větší části anomálie (chybné ohodnocení nebo normalizace).

7.2.2 Rozhodovací strom

Na trénovacích datech vykazuje tento algoritmus dobré výsledky, cca 93.8%, ale při náhodných pokusech nad alerty ze zcela "jiné" množiny (data z jiného roku), kde se vyskytlo hodně případů, které nebyly dostatečně podobné s trénovací množinou, klesla přesnost algoritmu na 27%. Všechny zbylé algoritmy se s tímto pokusem vypořádaly v tolerovaných hranicích. Z tohoto důvodu nebude rozhodovací strom v produkčním prostředí, ale pouze jako pomocný nástroj, což se předpokládalo už v kapitole 6.2.1.

Při sestavování trénovací množiny však pomocí DT dokážeme téměř okamžitě zjistit, že jsme např. zapomněli zahrnout / zvýraznit Temporary Internet Files příklady ze stanic, které i když nejsou smazané, tak se pro méně rizikový malware neeskaluje. Naopak např. Ransomware, který je nutné eskalovat i v případě smazání, neboť antivir obvykle maže jen soubor s instrukcemi pro platbu. Stejně tak zde chybí některé případy ze serverových infekcí. Produkční trénovací množina však bude sestavována až při nasazení, aktuálně stačí, že máme dvě techniky pro její kontrolu - chybné klasifikace a vizualizace přes DT. `RevoTreeView` pak dokáže vytvořit interaktivní webovou reprezentaci pro DT, vhodnou v případě rozsáhlejšího stromu.

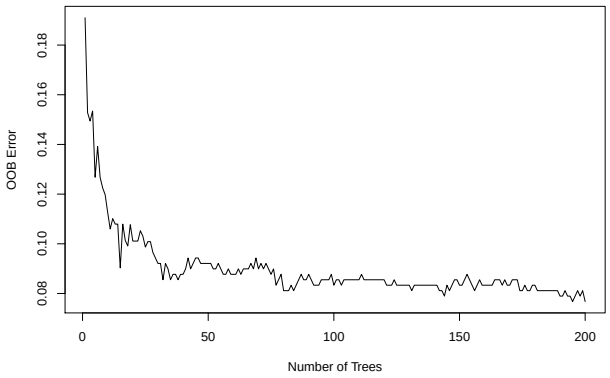
7.2.3 Random Forest

Obdobně jako u DT můžeme ovlivnit pomocí minSplit, minBucket a maxDepth to, jak moc se budou jednotlivé stromy rozrůstat (*min vzorků pro split, v listu a maximální hloubka stromu*). MinSplit je pro jiné než XDF formáty dat dle dokumentace doporučeno specifikovat ručně. Druhý zásadní parametr je, kolik stromů má daný RF mít. Algoritmus podporuje celou řadu dalších parametrů, které však nemají tak zásadní vliv na výsledek. Zde budeme hledat vhodné nastavení minSplit a nTrees. Vzhledem k výpočetní náročnosti nebudeme provádět brute-force hledání parametrů jako u SVM, pro odhad počtu stromů si pomůžeme OOB (Out-of-bag error), což je kumulovaná chyba jednotlivých klasifikátorů, viz obr. 41. Tato metrika existuje pouze pro Boosting a Bagging algoritmy, kde dochází k "sub-samplingu". Díky tomu je možno vypočítat chybu přímo na trénovacích datech, bez nutnosti testovací množiny.

Dle tabulky 3 je ideální minimální split 15 vzorků, graf na obrázku 41 pak potvrzuje, že pro tento split od 40 stromů výše je přesnost dostatečná.

RF		Počet stromů					
		20	35	50	75	100	150
Min. split	3	93,3%	93,3%	92,9%	92,9%	92,0%	93,8%
	10	91,6%	92,9%	91,6%	93,8%	93,8%	93,3%
	15	92,0%	92,0%	94,7%	93,3%	94,7%	94,2%
	20	92,4%	92,9%	92,9%	92,9%	92,0%	91,6%
	25	92,9%	93,3%	92,4%	93,3%	92,4%	94,2%

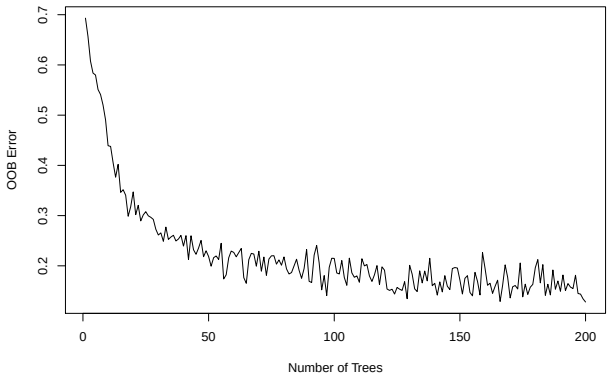
Tabulka 3: Vliv minSplit a nTree na přesnost RF



Obrázek 41: OOB chyba pro minSplit = 15

7.2.4 Boosted Decision Trees

Opět můžeme ovlivňovat růst stromu, jejich počet, ale také celou řadu dalších parametrů, ovlivňující výsledné chování. Názorně si zde vyzkoušíme vliv *learningRate* a minSplit, podobně jako u RF. Zatímco Random Forest vytváří plné klasifikační stromy, Boosting vede na větší množství stromů (viz 42), které nejsou moc hluboké (obvykle do 5-7 úrovní), což je ve výsledku méně výpočetně náročné než plné stromy u RF.



Obrázek 42: OOB error, výchozí parametry Boosted Decision Trees

Vliv learning rate a minSplit pro sto stromů, je pak zachycen v tabulce 4.

BDT		Learning rate					
		0,01	0,1	0,3	0,5	0,7	1
Min. split	3	85,8%	47,1%	91,1%	91,1%	90,7%	91,1%
	10	90,2%	88,9%	47,1%	82,2%	90,2%	89,8%
	15	91,1%	91,6%	88,0%	82,2%	90,2%	89,8%
	20	89,3%	91,6%	90,7%	85,8%	72,9%	91,6%
	25	88,0%	92,0%	89,8%	90,7%	85,8%	84,4%

Tabulka 4: Boosted Decision Trees, vliv learningRate a minSplit, nTree=100

7.2.5 Výkon a vliv duplicit

Pro srovnání výkonu jsou vybrány dva datasety, jeden je sestaven z dat, které jsou znormalizovány, ale počet instancí odpovídá zdrojovým datům (tj. obsahuje duplicity), druhý dataset je tvořen unikátními kombinacemi znormalizovaných dat. Oba datasety mají 10 kategoriálních atributů. Výchozí R balíčky pro rozhodovací stromy mají omezení na 32 kategorií v jednom atributu, RevoScale implementace tento limit teoreticky nemají, dají se navýšit parametricky, ale reálně při testu sto kategorií na atribut, nebyl schopen `rxDForest` tyto data zpracovat, zatímco `rxBTrees` s tím neměl žádný problém. To odpovídá způsobu konstrukce většího množství "mělkých" stromů, zmíněné v předchozí kapitole 7.2.4.

V tabulce 5 jsou vidět výsledky dvou testů výpočetního výkonu algoritmů, první množina je zatížena duplicitami. Pokud srovnáme výsledky s množinou obsahující unikátní kombinace, pak vidíme, že všechny algoritmy si dokáží s duplicitami poradit a nezvyšují jejich výpočetní náročnost.

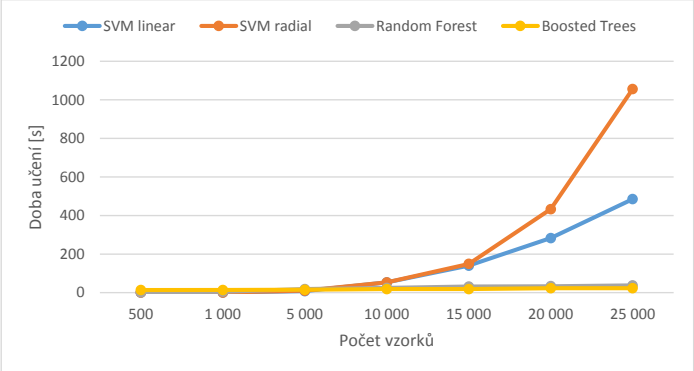
Algoritmus / doba učení [s]	Počet vzorků, data zatížená duplicitami ze 40-70%						
	500	1 000	5 000	10 000	25 000	50 000	100 000
SVM linear	0,04	0,18	5,8	30,61	150	757	1803
SVM radial	0,04	0,23	5,2	24,72	149	621	3063
Random Forest	8,6	9,56	12,44	13,68	27	22	28
Boosted Trees	6,36	6,81	7,39	8,11	10	14	20

Algoritmus / doba učení [s]	Počet vzorků, unikátní data						
	500	1 000	5 000	10 000	15 000	20 000	25 000
SVM linear	0,1	0,36	7,98	52,98	140	283	485
SVM radial	0,11	0,4	10,6	53,38	149	433	1056
Random Forest	1,44	4,94	18,81	24,9	32	34	38
Boosted Trees	13,54	13,87	14,98	18,55	19	23	23

Tabulka 5: Výkonnostní srovnání algoritmů a vliv duplicit

SVM však od většího počtu dat vyžaduje poměrně velký čas na natrénování modelu. Pokud bychom chtěli produkčně využít SVM na velké množině dat, pak dle [17, C.3] bylo otestováno převedení problému na detekci anomálií, kdy model tvoří data, které nevyžadují eskalaci a vše co nespadá do modelu (anomálie) budeme eskalovat. SVM však má více než výkonnostní

nevýhodu, především neznámé data. Nicméně pro ilustraci zvládá OneClassSVM 20tis. instancí za 8s, 77tis. za 200s. Prakticky došlo jen k posunutí hranice, kde se projeví složitost algoritmu, ale jde o poměrně citelné zlepšení. `rxOneClassSvm` by měl být dle dokumentace škálovatelný, což nebylo ověřeno (využívá však standardní knihovnu LIBSVM). Grafický průběh délky učení dle algoritmu na unikátních datech je zaznamenán na obrázku 43



Obrázek 43: Grafický průběh srovnání algoritmů na unikátních datech

7.2.6 Souhrn

Pokud se podíváme na pravděpodobnosti, jak jednotlivé algoritmy rozhodovaly o výsledné třídě (viz ilustrace na obr. 44), pak zde nalezneme řadu slabých klasifikací, kdy algoritmus rozhodne např. v poměru 55:45. Dále lze vidět, že se algoritmy rozhodují na základě stejných modelů rozdílně, příklad na řádku 5, kdy BT rozhodl s naprostou určitostí, že problém je vyřešen, pak RF dospěl k nepříliš jistému rozhodnutí eskalace, zatímco SVM eskaluje již téměř s jistotou. (Konkrétně tento případ je poměrně složitá situace, kdy dle detekce se jedná o high-risk malware ve formě .exe souboru, který byl karanténován, v reálu jde však o false-positive detekci na USB disku a je třeba vyžádat vzorek a zajistit nápravné akce.)

Pokud si shrneme současné poznatky, pak SVM vede v přesnosti klasifikace, ale nedovede se vypořádat s neznámými daty, kdy je třeba model přeučit na nové kategorie. To není problém pro menší trénovací množiny, ale od jisté velikosti modelu roste časová náročnost velmi strmě. Random Forest představuje "střední cestu", kdy konečné výsledky byly odhadnuty většinou správně, ale nemůže být využit pro atributy s velkým množstvím kategorií. Boosted Trees na zobecněné množině poskytoval horší výsledky, s detailnější množinou a velkým množstvím kategorií dosahoval úspěšnosti srovnatelnou s SVM.

Jako poslední pokus vyzkoušíme kombinaci klasifikátorů, konkrétně RF, který bude naučen na zobecněné množině a BT, který se naučí na detailnějších datech. Tím se pokusíme vyvážit slabší stránku BT, kterou je náchylnost k přeučení. Abychom nemuseli složitě hledat vztah, podle kterého vypočteme výsledné hlasování, tak si vypomůžeme jednoduchou neuronovou sítí. Vstu-

Filter

	Cleaned_prob	Escalate_prob	StatusName_Pred
1	0.005497384	0.99450262	Escalate
2	0.022421921	0.97757808	Escalate
3	0.003172939	0.99682706	Escalate
4	0.917794283	0.08220572	Cleaned
5	0.899998934	0.10000107	Cleaned
6	0.002544350	0.99745565	Escalate
7	0.002791296	0.99720870	Escalate
8	0.000024105	0.99997590	Escalate
9	0.967503124	0.03249688	Cleaned
10	0.967503124	0.03249688	Cleaned
11	0.967503124	0.03249688	Cleaned
12	0.434247992	0.56575201	Escalate
13	0.000353148	0.99964685	Escalate

(a) Boosted Trees

Filter

	Cleaned_prob	Escalate_prob	StatusName_Pred
1	0.09418	0.90582	Escalate
2	0.31430	0.68570	Escalate
3	0.01818	0.98182	Escalate
4	0.83480	0.16520	Cleaned
5	0.42570	0.57430	Escalate
6	0.07381	0.92619	Escalate
7	0.05997	0.94003	Escalate
8	0.01899	0.98101	Escalate
9	0.76592	0.23408	Cleaned
10	0.76592	0.23408	Cleaned
11	0.76592	0.23408	Cleaned
12	0.44284	0.55716	Escalate
13	0.02628	0.97372	Escalate

(b) Random Forest

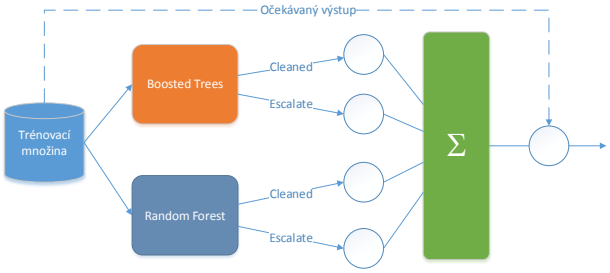
Filter

Cleaned	Escalate
0.070913	0.92909
0.214272	0.78573
0.117565	0.88243
0.876626	0.12337
0.252049	0.74795
0.127766	0.87223
0.048420	0.95158
0.127732	0.87227
0.876513	0.12349
0.876513	0.12349
0.876513	0.12349
0.312244	0.68776
0.015902	0.98410

(c) SVM

Obrázek 44: Pravděpodobnostní příslušnost do kategorie

pem sítě jsou pravděpodobnosti jednotlivých kategorií, tj. čtyři neurony, skrytá vrstva dvanáct a výstup představuje jeden neuron. Síť natrénujeme pomocí získaných pravděpodobností jednotlivých klasifikátorů, vůči očekávané klasifikaci z trénovací množiny.



Obrázek 45: Trénování kombinovaného klasifikátoru

Výsledná klasifikace je uvedena v tabulce 6. Na prvním řádku testujeme pracovní klasifikaci, kdy hledáme anomálie. Druhý řádek obsahuje test nad certifikovanou množinou, která je rozdělena v poměru 70:30. Další dva řádky jsou náhodné výběry necertifikovaných dat, které obsahují i chybné rozhodnutí.

Trénovací množina / testovací množina	Velikost test mn.	Max 32 kategorií / atribut			Včetně detailních kategorií		
		SVM	B-Trees	R-Forest	SVM	B-Trees	BT+RF=>NN
Train / Train	700	98,23	93,81	95,29	99,55	98,67	98,52
Train / Test [70:30]	**210**	**89,77**	**85,77**	**92,00**	**92,44**	**92,89**	**92,89**
Train / Random za 3 roky	7 000	65,15	61,73	54,86	76,28	68,90	75,92
Train / 10. týden 2017	16 500	91,18	83,35	86,73	91,10	85,70	85,99

Tabulka 6: Celková klasifikace

7.3 Model, jeho aktualizace a rozhraní pro klasifikaci alertů

Model se vytváří z množiny alertů, které byly označeny oprávněným specialistou jako `Certified` v monitorovacím GUI. Pro tvorbu a aktualizaci modelu máme na výběr dvě varianty:

- Vytvoření modelu na lokálním počítači a následný deployment na server.
- Automatické vytvoření modelu v rámci SQL uložené procedury.

První varianta je ideální pro případy, kdy je frekvence aktualizace modelu nízká. V našem případě však využijeme variantu dvě, kdy se model nechá přebudovat každou noc na základě nových dat. Persistence modelu je zajištěna serializací přes metodu `serialize(model)` a následnému uložení do tabulky modelů, datový typ `varbinary(max)`. V tabulce jsou uchovány i předchozí verze, pro predikci je pak využita nejnovější, kterou má obsluha v případě defektu možnost jednoduše odstranit, čímž se automaticky přejde na starší, ověřený model.

Praktický postup by vypadal následovně. Nejprve si v IDE připravíme R skript, který natrénujeme na zdroji např. z CSV (viz výpis 9). Následně upravíme zdroj na `InputDataSet`, což je proměnná, přes kterou přijmeme při volání přes SQL proceduru vstupní data.

```
require(RevoScaleR)
...
trainData <- InputDataSet
rxBTreesFormula <- StatusName ~ CustomerSecurityAwareness + RiskAssesment + FilePathCategory + FileCategory + ActionName + MalwareType + ...
trainedModel <- rxBTrees(rxBTreesFormula, data = trainData, nTree = 100, importance = TRUE, lossFunction = "multinomial", ...)
rxBTreesModel <- data.frame(model=as.raw(serialize(trainedModel, NULL)));
```

Výpis 9: Příprava R scriptu pro vytvoření modelu.

Stejným způsobem si připravíme SQL dotaz pro získání normalizovaných dat. Oba kódy překopírujeme do textových proměnných, které využíváme při volání externího skriptu (viz výpis 10). Výsledný model uložíme do tabulky modelů. U predikce pak postupujeme obráceně, pomocí @params vložíme natrénované modely a přes @input_data vstup pro klasifikaci.

```
create procedure [Antivirus].[uspTrainBTrees] as
begin
    declare @getTrainDataQuery nvarchar(max) = N'select ...normalize... from Antivirus.Alert where ...';
    declare @trainBTreesScript nvarchar(max) = N'
        rxBTreesFormula <- StatusName ~ FilePathCategory + FileCategory + ActionName + RiskAssesment + MalwareType + ...
        trainedModel <- rxBTrees(rxBTreesFormula, data = InputDataSet, nTree = 100, importance = TRUE, lossFunction = "multinomial", ...)
        rxBTreesModel <- serialize(trainedModel, NULL)';

    declare @model varbinary(max); -- pro slozeny klasifikator definujeme promennou pro kazdy model

    exec sp_execute_external_script
        @language = N'R',
        @script = @trainBTreesScript,
        @input_data_1 = @getTrainDataQuery,
        @params = N'@rxBTreesModel varbinary(max) OUTPUT', -- pro slozeny klasifikator definujeme promennou pro kazdy model
        @rxBTreesModel = @model OUTPUT -- pro slozeny klasifikator definujeme promennou pro kazdy model

    insert into Antivirus.ClassificationModel(Name, Model, CreatedAtUtc)
    values ('BTrees', @model, GetUtcDate());
end;
```

Výpis 10: Tvorba modelu v rámci SQL procedury.

8 Závěr

V úvodu práce byl čtenář seznámen se základní činností Tieto Security týmu, klíčovými pojmy z řešené oblasti a principy správy antivirů. Na základě nevyhovujícího aktuálního stavu byly analyzovány požadavky, ze kterých vznikl rámcový návrh architektury systému.

Ten byl dále rozveden v kapitole věnující se implementaci části systému, která nevyužívá oblast strojového učení ani automatizované podpoře lidské činnosti. Byla navržena a implementována kolekce služeb, která zajišťuje přenos dat a příkazů přes různé typy kanálů se spojově i nespojově orientovanou komunikací. Dále byla implementována desktopová GUI aplikace sloužící k administraci a monitorování provozovaných služeb, jejíž architektura byla navržena v rámci bakalářské práce. Všechny zmíněné aplikace jsou řešeny jako modulární systémy s volnou vazbou.

V následující kapitole byl v detailech představen proces zpracování alertu a identifikace oblastí pro automatizaci. Byly popsány algoritmy strojového učení, které byly využity pro tvorbu trénovací množiny a produkční klasifikátor. Dále byly popsány techniky, jak pomocí kombinace pravidlových systémů a šablon docílit automatického sestavení instrukcí a jejich eskalaci na zodpovědné osoby. Pro tento účel byla pomocí ANTLR navržena vlastní gramatika a parser pro vyhodnocení těchto pravidel.

Navržený systém je schopen redukovat množství alertů vstupujících k lidské analýze o více než 95% a následně snížit čas pro sepsání instrukcí, vyhledání zodpovědné osoby a eskalaci do cílového systému na naprosté minimum, což představuje roční úsporu v řádu mnoha tisíc hodin.

Tento stav však není cílový. Při produkčním nasazení se začne vytvářet trénovací množina z alertů, které bylo možné bez úprav odeslat s navrženým řešením na navržené kontakty. Stejnými algoritmy a postupy, na kterých je postaven tento systém, pak dosáhneme stavu, kdy nebude potřeba téměř žádné opakované manuální činnosti a systém bude využívat znalostí předaných analytikem pro nové problémy.

V době odevzdání této práce byla ověřena i rozšiřitelnost implementovaného systému, na kterém již běží monitoring SIEM a IPS. Dále řada integrací pro automatizovaný reporting spravovaných systémů, které nevyžadují monitorování.

Literatura

[1] Malware Statistics & Trends Report. *AV TEST* [online]. Dostupné z: `https://www.av-test.org/en/statistics/malware/`

[2] STEWART, James Michael, Ed TITTEL a Mike CHAPPLE. *CISSP: Certified Information Systems Security Professional Study Guide.* 5th edition. Indianapolis, Indiana: Wiley Publishing, 2011. ISBN 978-0470944981.

[3] PCI SECURITY. *PCI SECURITY* [online]. Dostupné z: `https://www.pcisecuritystandards.org/pci_security/`

[4] SEEMANN, Mark. *Dependency injection in .NET.* Shelter Island, NY : Manning, 2012. str. 552. ISBN 19-351-8250-1.

[5] BRUMFIELD, Bob.*Developer's guide to Microsoft Prism 4: building modular MVVM applications.* Redmond, Wash. : Microsoft, 2011. p. 261. ISBN 978-0-73565-610-9.

[6] NATHAN, Adam. *WPF 4 unleashed.* Indianapolis, Ind. : Sams, 2010. p. 825. ISBN 06-723-3119-5.

[7] Managed Extensibility Framework. *MEF* [online]. Dostupné z: `https://msdn.microsoft.com/en-us/library/dd460648(v=vs.110).aspx`

[8] WiX Toolset. *WiX Toolse* [online]. Dostupné z: `http://wixtoolset.org/`

[9] HOHPE, Gregor. a Bobby. WOOLF. *Enterprise integration patterns: designing, building, and deploying messaging solutions.* Boston: Addison-Wesley, c2004. ISBN 978-0321200686.

[10] WITTEN, I. H., Eibe. FRANK a Mark A. HALL. *Data mining: practical machine learning tools and techniques. 3rd ed.* Burlington, MA: Morgan Kaufmann, c2011. Morgan Kaufmann series in data management systems. ISBN 978-0-12-374856-0.

[11] About Microsoft R. *Microsoft R* [online]. Dostupné z: `https://msdn.microsoft.com/en-us/microsoft-r/microsoft-r-getting-started`

[12] Aggarwal, Charu C. *Data Mining: The Textbook* Springer International Publishing Switzerland 2015. ISBN 978-3-319-14142-8.

[13] KELLEHER, John D., Brian MAC NAMEE a Aoife D'ARCY. *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies.* ISBN 978-0-262-02944-5.

[14] DUDA, Richard O., Peter E. HART a David G. STORK. *Pattern classification.* 2nd ed. New York: Wiley, c2001. ISBN 978-0471056690.

[15] ROKACH, Lior a Oded MAIMON. *Data mining with decision trees theory and applications*. Singapore: World Scientific, 2008. ISBN 9789812771728.

[16] BISHOP, Christopher M. *Pattern recognition and machine learning*. New York: Springer, c2006. Information science and statistics. ISBN 978-0387-31073-2.

[17] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. *A Practical Guide to Support Vector Classification*. National Taiwan University [online]. 2003, 16 [cit. 2017-04-06]. Dostupné z: `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`

[18] PARR, Terence. *The definitive ANTLR 4 reference*. Pragmatic programmers. ISBN 978-1-93435-699-9.

[19] Microsoft. *Prism 5.0 for WPF* [online]. Dostupné z: `https://msdn.microsoft.com/en-us/library/gg406140.aspx`